

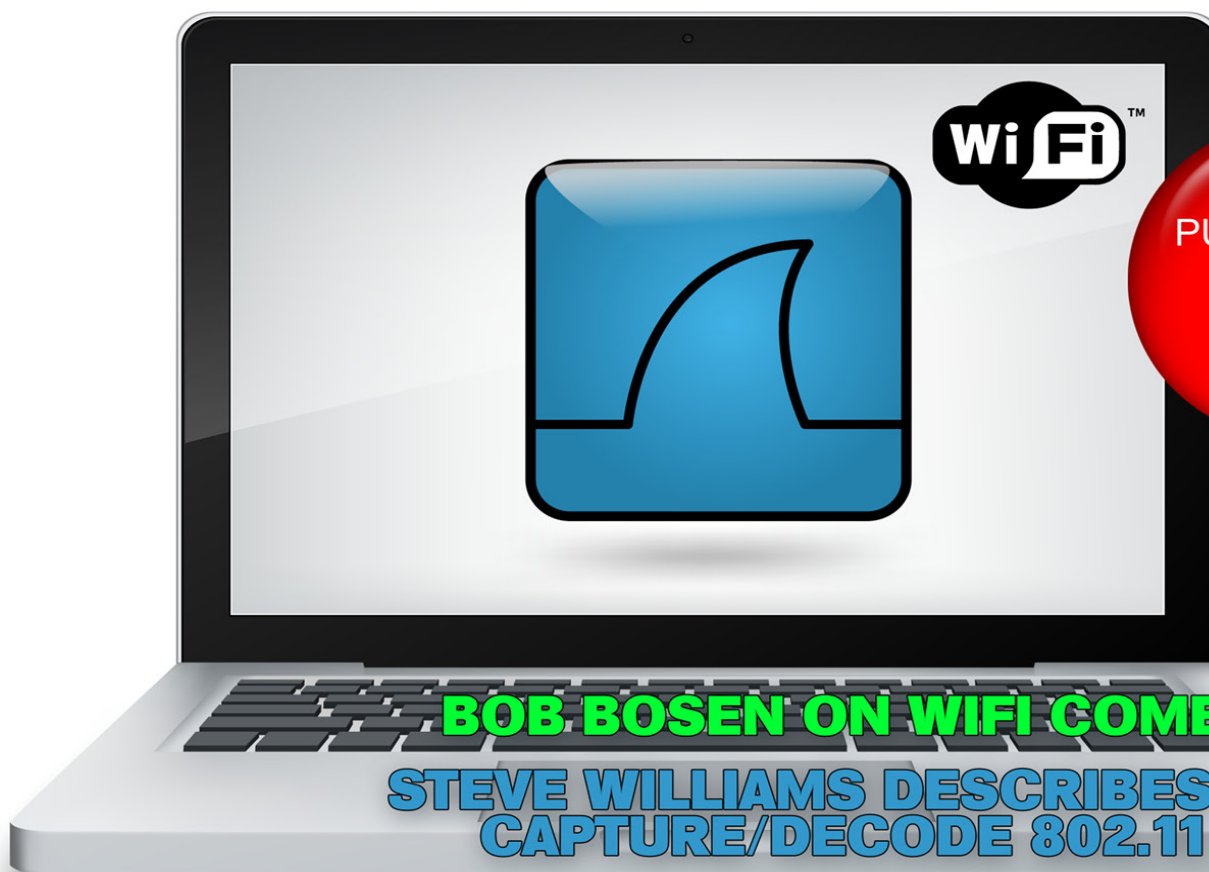
HAKING

Vol.1 No.7
Issue 07/2012(7) ISSN: 1733-7186

ON DEMAND



WIRESHARK SHARKS ON THE WIRE



SPECIAL
PUBLICATION
70+
PAGES

BOB BOSEN ON WIFI COMBAT ZONE

**STEVE WILLIAMS DESCRIBES HOW TO
CAPTURE/DECODE 802.11 TRAFFIC**

**WILLIAM F. SLATTER III SHOWS HOW TO SOLVE
"ATTRIBUTION PROBLEM"**

**HAI LI DISCUSSES ANALYZING
A WIRELESS PROTOCOL**

PLUS

LEARN MORE ABOUT WIRESHARK!
PEDRO MORENO SANCHEZ AND ROGELIO MARTINEZ PEREZ
SHOWS HOW TO USE COOJA SIMULATOR TOGETHER
WITH WIRESHARK

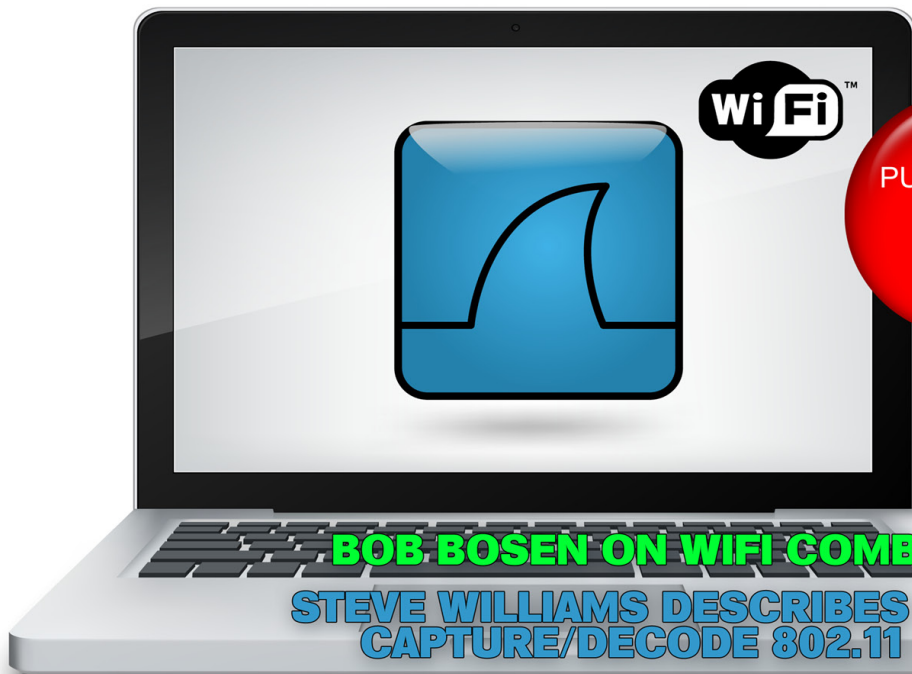
HAKING

Vol.1 No.7
Issue 07/2012(7) ISSN: 1733-7186

ON DEMAND



WIRESHARK SHARKS ON THE WIRE



SPECIAL
PUBLICATION
70+
PAGES

BOB BOSEN ON WIFI COMBAT ZONE

**STEVE WILLIAMS DESCRIBES HOW TO
CAPTURE/DECODE 802.11 TRAFFIC**

**WILLIAM F. SLATTER III SHOWS HOW TO SOLVE
"ATTRIBUTION PROBLEM"**

**HAI LI DISCUSSES ANALYZING
A WIRELESS PROTOCOL**

PLUS

LEARN MORE ABOUT WIRESHARK!

**PEDRO MORENO SANCHEZ AND ROGELIO MARTINEZ PEREZ
SHOWS HOW TO USE COOJA SIMULATOR TOGETHER
WITH WIRESHARK**



11th International Conference on Software QA and Testing on Embedded Systems

MORE TESTING

This year we offer a special program: **3 Keynotes, 2 Tutorials, 8 Tracks**, from renowned companies such as: **Intel, Samsung, Phillips, Adobe, Oracle and Quality Perspectives.**

MORE QUALITY

QA&TEST, the international Conference on Software QA and Testing on Embedded Systems, will be held on 17, 18 and 19 October in Bilbao, Spain, and gives you an excellent opportunity to increase your business network and establish contact with others professionals of the industry.

MORE EMBEDDED

The main objective of QA&TEST is present the last technological developments in Software Testing and Quality Assurance, and showcase successful best practice to reduce costs and give companies a lead in global competition.

SOFTWARE

www.qatest.org

Special offer 15% discount using the code **HAKIN9**

Organiser



Sponsor



October 17 · 18 · 19

2012

Bilbao · Spain



OWASP AppSec USA 2012 Hyatt Regency, Austin TX

Training (October 23-24, 2012)

\$750 for 1-day courses

\$1500 for 2-day courses

Conference (October 25-26, 2012)

\$595 (10% off with HACKIN9 code)

www.appsecusa.org

Are you aware of the latest in Application Security?

OWASP AppSec conferences bring together industry, government, security researchers, and practitioners to discuss the state of the art in application security.

This series was launched in the United States in 2004 and Europe in 2005.

Global AppSec conferences are held annually in North America, Latin America, Europe, and Asia Pacific.



SCAN THE QR
FOR MOBILE
REGISTRATION!

KEYNOTE SPEAKERS:



Douglas Crockford
Architect, PayPal



Michael Howard
Principal
Cybersecurity
Architect at
Microsoft



Gene Kim
Researcher,
TripWire Founder

HACKIN9
ON DEMAND

ewa.dudzic@hakin9.org

Managing Editor: Ewa Duranc

ewa.duranc@hakin9.org

Paweł Płocki

pawel.plocki@hakin9.org

Editorial Advisory Board: Board: Arsen Darakdjian, Scott Paddock, Matthew Holley, Derek Thomas, Ewelina Soltysiak

Proofreaders: Ewa Duranc, Arsen Darakdjian, Scott Paddock

Special Thanks to the Beta testers and Proofreaders who helped us with this issue. Without their assistance there would not be a Hakin9 magazine.

Senior Consultant/Publisher: Paweł Marciniak

CEO: Ewa Dudzic

ewa.dudzic@hakin9.org

Production Director: Andrzej Kuca

andrzej.kuca@hakin9.org

Art Director: Ireneusz Pogroszewski

ireneusz.pogroszewski@hakin9.org

DTP: Ireneusz Pogroszewski

Marketing Director: Paweł Płocki

pawel.plocki@software.com.pl

Publisher: Software Press Sp. z o.o. SK

02-682 Warszawa, ul. Bokserska 1


Phone: 1 917 338 3631

www.hakin9.org/en

Whilst every effort has been made to ensure the high quality of the magazine, the editors make no warranty, express or implied, concerning the results of content usage.

All trade marks presented in the magazine were used only for informative purposes.

All rights to trade marks presented in the magazine are reserved by the companies which own them.

To create graphs and diagrams we used  program by 

Mathematical formulas created by Design Science MathType™

DISCLAIMER!

The techniques described in our articles may only be used in private, local networks. The editors hold no responsibility for misuse of the presented techniques or consequent data loss.

Dear Hakin9 followers,

This month's issue is devoted to Wireshark. We have many articles that were written especially for you by network security experts. The first article is by Patrick Mark Preuss who describes network related problems and proves that Wireshark is the right tool to do that. What is more, we have an article by Bob Bosen who shows that there are several means by which your neighbors could have penetrated your WiFi LAN. Next, Hai Li demonstrates how to support a new wireless protocol in Wireshark. Also, we have an article by William Favre Slater III on Cyberwarfare and Cybercrime. Last but not least, you should take a look at article by Pedro Moreno-Sanchez and Rogelio Martinez-Perez. Their article is devoted to tracing ContikiOS based IoT communications over Cooja simulations with Wireshark.

Hakin9's editorial team would like to give special thanks to the authors, betatesters, proofreaders and our editor in chief, Ewa Dudzic.

I hope that you will enjoy reading this issue!

Ewa Duranc, Paweł Płocki & the Hakin9 Team.

A message from the whole team

It has been a difficult time for us over the past few weeks. A mistake has been made on our part, which has led to a vast amount of criticism towards Hakin9. This situation has influenced us to reflect on our past choices and policies. We want to assure you that from now on we will be working even harder to bring you the best material on IT security out there.

We also wanted to take this time to thank all of you for staying by our side; with special thanks to our authors, beta-testers, proofreaders and partners.

Thank you all for your support. We are truly blessed to be able to work with such gifted individuals and to have such amazing readers.

Hakin9 Magazine's Editorial Team



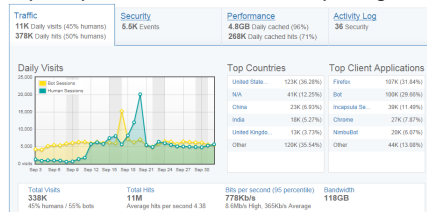
If you own or manage a website, two of your biggest concerns are keeping it secure and making it faster. Thanks to [Incapsula](#), a cloud-based service for SME's, these needs can be fully met with a simple setup and without installing any hardware or software. [Incapsula](#) protects and accelerates websites. The service includes a bot detection technology to identify and filter out malicious bots, a PCI-certified Web Application Firewall (WAF) to provide Enterprise-grade website security against malicious attacks and hackers, and a content delivery network (CDN) to accelerate websites.

How does it work?

Once the user signs up to Incapsula, he gets simple directions to change his website's DNS records in order to route the traffic through Incapsula's global network of data centers. After the DNS changes have been completed (it takes minutes to make the changes and up to 24 hours for the changes to fully propagate - but there is no downtime during the process), the incoming traffic is profiled in real-time by Incapsula, allowing Incapsula to block all threats to the website. Meanwhile, the outgoing traffic is accelerated and optimized by Incapsula.



By offering a distributed, cloud-based service, Incapsula is able to offer a variety of affordable plans, starting with a free plan for small websites and up to Business and Enterprise plans which include Enterprise-grade security features.



The Dashboard

The Traffic tab provides an overview of the website visits and sessions, separating human visits and bot visits, along with a list of top countries & Applications used to hit the website.

The Security tab lists the different threats and incidents identified by Incapsula. For each threat there is an option to view the event - which means to get a detailed session report. You can also view your current setting - is Incapsula blocking the threat, reporting it or completely ignoring it.

Threat Type	Incidents	Current Setting	
Visitors from blacklisted IPs	0	1 IP in blacklist	View Events
Visitors from blacklisted Countries	N/A	No Countries in blacklist	Add Countries
Bad Bots	1.3K	Block	View Events
Suspected Bots	N/A	Ignore	Enable
SQL Injection	4	Block	View Events
Cross Site Scripting	2	Block	View Events
Illegal Resource Access	11	Block	View Events
DDoS	0	Protected	View Events

The *Performance* tab in the dashboard helps in ascertaining the bandwidth consumption and also provides insight on how much speed has been increased by the caching and optimization features provided by Incapsula.



As Incapsula provides a global Content delivery network (CDN), by presenting the content of the website to the visitor from the nearest data center, the pages load can be dramatically improved, which improves SEO ranking and of course the user experience.

Settings

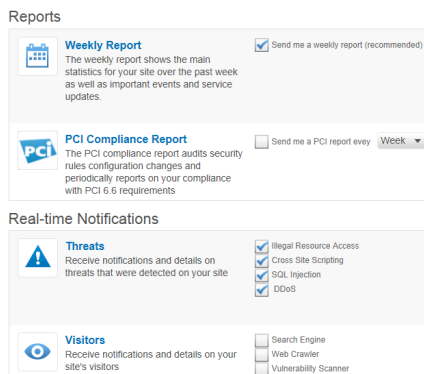
Incapsula offers various setting options to allow the user full control of his security and performance rules:



The Site Settings screen offers various options but the most interesting one is the 'Advanced Acceleration Mode' that enables Incapsula's unique dynamic content caching. In my case, indeed it improved site performance significantly.

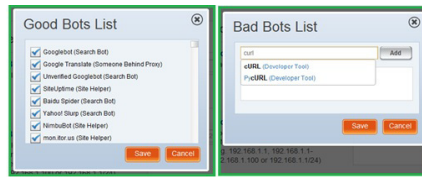
The *Notifications Settings* provides a few options to make sure that the user is always updated with the latest events:

- A Weekly report: sent automatically to the email, listing all important security events in the passing week, and other traffic and performance stats
- A PCI Compliance Report: PCI compliance is an important issue for e-commerce sites. The PCI report is a unique feature of Incapsula and allows you to provide your auditor with a report indicating that your website has been protected by a Web Application Firewall during a specific period.
- Real-time Notifications: Including notification emails for specific types of threats and visitors to the site.

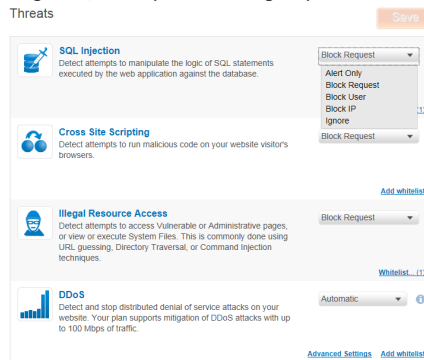


The *Security Settings*: Incapsula's Bot Access Control allows the user to decide which bots are allowed to visit the website, which should be blocked, and which should get a CAPTCHA test.

As some of the bots disguise themselves through spoofed IPs & fake user-agents, Incapsula uses various identification techniques, to search for clues in the behavior patterns of the bot and the HTTP Headers.

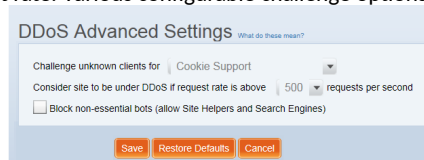


The WAF Settings: Incapsula's PCI-Compliant Web Application Firewall protects from 4 types of threats: SQL Injection, XSS, Illegal Resource Access and DDoS attacks. For each type of attacks the user has various options, starting from deciding what Incapsula should do with the attack (Alert only, Block, ignore) and up to Blocking a specific User or IP.



DDoS Settings

The 'Automatic' feature in DDoS gets enabled at the time of attack. The 'Advanced Settings' option allows site administrator to configure the challenge and DDoS request rate. Various configurable challenge options are Cookie / Javascript / Captcha support.



Overall Assessment:

My overall assessment of the tool is 9 on a scale of 1-10 and would recommend customers using Incapsula for protecting their websites against web threats and accelerating it.

If you have an eCommerce website, Incapsula provides a PCI-Certified WAF with excellent reports that can be filed in your compliance file. If you are a customer to whom true security, confidentiality, integrity, and availability are of crucial or of utmost importance - this is, in my opinion, one of the best tools to help you to reach this goal.

<http://www.incapsula.com/>



Amit Chugh

Amit Chugh, CEH®, ISO 27001 LA, Creative, innovative & results driven technology leader with over 14 years of industry experience, with 7+ years specialising in Information Security Management, Incident Management, Business Continuity Management and Software Development. He is reachable at chugh 'dot' a 'at' gmail 'dot' com.

Wireshark – Sharks on the wire

Capturing and analyzing network data is one of the core skills every IT professional should possess. If you have problems with your system or application, suspect a security issue, in almost every case the network is involved today. Wireshark is the right tool to help you finding network related problems and analyze them.

Wireshark can be used for different tasks: Troubleshooting network problems, security analysis, optimization, and application analysis. Network

data analysis can be a huge field and can be confusing if you are not so familiar with it.

History

Before we begin with the Wireshark itself, we should have a look into the history of packet tracing. Programs for network tracing are known since the late 1980's. At that time mainly commercial analyzers were unavailable, the most famous being at this time was the program *Sniffer*, developed by Network General. You may have noticed that the process, is sometimes called *sniffing*, this term goes back to this program. On Unix machines the program *tcpdump* has been developed by Van Jacobsen, Leers and MacCanne in the late 1980s, this program and the library *libpcap* can be seen as the grand fathers of Wireshark. In the early 1990s there were a lot of commercial packet analyzers available, most of them was expensive and built in hardware. This changed at the end of the 1990s with the development of "Ethereal" by Gerald Combs, this program was build on top of *libpcap* and the *GIMP Tool Kit* (GTK) library, this brought a free analyzer to many different operating systems. In 2006 Gerald Combs changed employment to *CASE Technologies* and new project was started on the code base from *Ethereal*. The program since than is called *Wireshark*. Wireshark is available on many different platforms, for example Microsoft Windows, Linux/Unix and OSX, it can now be seen as the standard application for network analysis.

TCP/IP Basics

Wireshark can deal with a many protocols families. To name some there are AppleTalk, wireless protocols like *Wlan*, *WiMax* and the famous *TCP/IP*. We should have a look on TCP/IP protocol suite because it is the most frequently used protocol today.

The protocol was developed by the *Defense Advanced Research Projects Agency* (DARPA) in the 1970s, its roots go back to the ARPANET (*Advanced Research Projects Agency Network*).

TCP/IP provides end-to-end connectivity, specify how data should be formatted, addressed, transported and routed.

The suite is divided into four layers, each with its own set of protocols, from the lowest to the highest:

The physical layer defines wiring, electrics and low level protocols to access the media and address nodes on the same medium. As an example can be seen: Ethernet, Wireless, DSL (*Digital Subscriber Line*), PPP (*Point to Point Protocol*) and others. The addresses used on this layer are called MAC Address.

The *internet layer* (IP) is for addressing the nodes: each node becomes a global unique address. The addressing can be IPv4 or IPv6. IPv4 addresses are usually written as dotted decimal numbers, for example, 192.168.0.1.

The protocol has an address space of 32bit = $2^{32} = 4.294.967.296$ and this space cannot give every device on the plant an address. To overcome this, there is a technique called *Network Address Translation* (NAT).

To address this issue in 1998, the Internet engineering task force (IETF) has released a new protocol standard to solve this problem. This protocol standard is called IPv6 and brings many improvements over IPv4, such as: a bigger address space, encryption support (ipsec), and has been redesigned so that new feature can be easily implemented. The Addresses are now 128 bit long and will provide $3.403 \times 10^{38} = 2^{128}$ unique addresses.

Routing is used when addresses are not local in your network. Most systems have a default route to a router, which can forward these packets. There is no magic in it, any system knows its own IP address and the network mask, for example, the address is 192.168.0.100, and the network mask is 255.255.255.0. Netmask can also be written in another format, CIDR (Classless Inter-Domain Routing). Here netmask will be written /24, which means that the first 24 bits from the address are the network and the remaining bits are the node. With this notation, it is obvious that the host 10.0.0.1 is not on the same network and that the packets need to be send to the router.

The transport layer defines how data will be transported. *Transmission Control Protocol* (TCP) is used for reliable transport of the data, like file transfer or email. On the other hand, there is *User Datagram Protocol* (UDP), with which the data sent is unreliable, and is used for time critical applications like VoIP (Voice over IP). These applications have the need of continuous arrival of packets and the information stored in a single packet is not so important.

The Application Layer defines how the data is encoded, for example, HTTP (Hyper Text Transfer Protocol), SMTP (*Simple Mail Transfer Protocol*), SIP (Session Initiator Protocol – VoIP Call Control Protocol).

In the Table 1 you will find an overview of the TCP/IP suite.

Table 1. TCP/IP Layers

OSI Layer	TCP/IP Layer	Example
Application (7)		
Presentation (6)	Application	HTTP, SMTP, POP, SIP
Session (5)		
Transport (4)	Transport	TCP, UDP, SCTP
Network (3)	Internet	IP (IPv4,IPv6)
Data Link (2)		
Physical (1)	Link	Ethernet, Wireless, DSL

When you are not so familiar with the tcp/ip you can use Wireshark to expand your knowledge. For example, you can trace the packets when opening the URL <http://www.wireshark.org> in a web browser and see what happens. You will see that the name is translated with DNS (*Domain Name Service*) to an IP address and then, a TCP session to the address is opened. Note: Please be aware when firewalls or WAN optimizers are installed in the path, they can alter TCP/IP behavior and packet contents.

Getting started with captures

Getting started with data capture with Wireshark is pretty easy. The program installs all the necessary components for capturing data. Wireshark comes with an easy-to-use interface, many analysis features and tools. When you start Wireshark, you will see the main window. Here you can select the interface which should be used for data capture. During the capture, you will see a live packet list and an analysis (Figure 1).

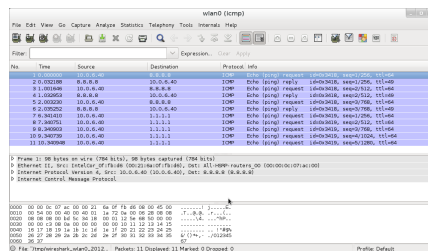


Figure 1. Capture window

What we see during a sample capture is that there was a ping to www.wireshark.org and the answers. It is also possible to use Wireshark from the command line (Listing 1). First, we looked up the available interfaces with `tshark -D` and then, we started a capture on `tshark -i wwan0`, in (Table 2) you can see some of the common command line options.

Listing 1. Command line usage

```
[~]# tshark -D
1. eth0
2. eth1
3. any (Pseudo-device that captures on all interfaces)
4. lo
[~]# tshark -i eth0
Capturing on eth0
1.121921 10.0.12.10 -> 174.137.42.75 ICMP 98 Echo (ping) request id=0x03f9, seq=1/256,
    ttl=64
1.307740 174.137.42.75 -> 10.0.12.10 ICMP 98 Echo (ping) reply id=0x03f9, seq=1/256,
    ttl=51
2.122759 10.0.12.10 -> 174.137.42.75 ICMP 98 Echo (ping) request id=0x03f9, seq=2/512,
    ttl=64
2.305570 174.137.42.75 -> 10.0.12.10 ICMP 98 Echo (ping) reply id=0x03f9, seq=2/512,
    ttl=51
3.123583 10.0.12.10 -> 174.137.42.75 ICMP 98 Echo (ping) request id=0x03f9, seq=3/768,
    ttl=64
3.307118 174.137.42.75 -> 10.0.12.10 ICMP 98 Echo (ping) reply id=0x03f9, seq=3/768,
    ttl=51
6 packets captured
[~]#
```

Listing 2. Using multiple files

```
[~]$tshark -i eth1 -w /tmp/out.pcap -b duration:2 host www.Wireshark.org
Capturing on eth1
108
[~]$ls -la /tmp/out*
-rw----- . 1 root root 176 Oct 3 20:11 /tmp/out_00001_20121005201159.pcap
-rw----- . 1 root root 28084 Oct 3 20:12 /tmp/out_00002_20121005201201.pcap
-rw----- . 1 root root 16568 Oct 3 20:12 /tmp/out_00003_20121005201203.pcap
-rw----- . 1 root root 21396 Oct 3 20:12 /tmp/out_00004_20121005201205.pcap
-rw----- . 1 root root 176 Oct 3 20:12 /tmp/out_00005_20121005201207.pcap
```

In the GUI, you have the option to save the data to a file after you have captured it, or during the setting up a new capture. It is possible to use more than one file. This is useful when capturing high volume of traffic or switch files on a regular base. My personal favorite for capture is the command line because less system resources are used and you can easily use it on remote systems. Listing 2 shows how it looks when using multiple files.

Table 2. Tshark options

-i <interface>	name or idx of interface (def: first non-loopback)
-D	print list of interfaces and exit
-n	disable all name resolutions (def: all enabled)
-w <outfile>	write packets to a pcap-format file named „outfile”filesize:NUM – switch to next file after NUM KB
-b <capture ring buffer option>	filesize:NUM – switch to next file in NUM KB duration:NUM – switch to next file in NUM seconds
-r <infile>	set the filename to read from (no pipes or stdin!)
-Ttext fields	format of text output
-e <field>	field to print if -Tfields selected (e.g. tcp.port); this option can be repeated to print multiple fields
-R <read filter>	packet filter in Wireshark display filter syntax

The needle in a haystack

So far we have seen how to capture data, but we might see a lot of data. To get useful information out of huge captures might not be easy, it's like trying to find the needle in a haystack. Wireshark can help us to limit the traffic we capture and see. There are two type of filters: *capture filters* are used during the capture process and are applied directly to the interface. This will use less system's resources, they are a good starting point to reduce the amount of traffic we capture. Some examples: to filter traffic to a particular host: `host 192.168.0.1`, a network `net 192.168.0.0/24` or a specific application like HTTP `port 80`

When you are beginning a new capture, the filter can be applied directly on the command line or in the capture options dialog, for example: `tshark -i eth0 host www.wireshark.org` this will capture all the traffic from and to www.wireshark.org.

There are more options if you have to write filters, for more details please use the Wireshark Wiki and the *libpcap* site. Capture filters are implemented in the library. The same filters can be used with any pcap based program like `tcpdump`. You can use those filters, for example, for security analysis, like this one for the blaster worm `dst port 135 and tcp port 135 and ip[2:2]==48`. The display filters, on the other hand, give access to the processed protocols, the filter can be used also during the capture or after the capture has been finished. For example, `tcp.analysis.ack_rtt` gives you access to the acknowledgment round trip times, Hosts can be selected with `ip.host eq <hostname>` OR `ip.src, ip.dst`. The filters are powerful tool for limiting the display of the captured packets. You have the possibility to look for errors, follow specific streams or see which urls have been accessed, you can even trace SIP Calls and look for a specific number. For example: `http.request.uri contains "GET"`. In listing 3 you can see an example capture to [Wireshark.org](http://www.wireshark.org) in the first part we have used a capture filter we will see the complete tcp traffic, tree-way handshake and the GET request for the [Wireshark](http://www.wireshark.org) homepage. In the second part, we applied a display filter that shows us only the GET request for the homepage.

Listing 3. Capture and display filters

```
[~]$tshark -i eth0 host www.Wireshark.org
Capturing on eth0
0.000000 10.0.12.10 -> 174.137.42.75 TCP 74 48739 > http [SYN] Seq=0 Win=14600 Len=0
MSS=1460 SACK_PERM=1 TSval=70646065 TSecr=0 WS=16
0.184523 174.137.42.75 -> 10.0.12.10 TCP 74 http > 48739 [SYN, ACK] Seq=0 Ack=1
Win=5792 Len=0 MSS=1452 SACK_PERM=1 TSval=641801134 TSecr=70646065 WS=128
```

```

0.184598 10.0.12.10 -> 174.137.42.75 TCP 66 48739 > http [ACK] Seq=1 Ack=1 Win=14608
      Len=0 TSval=70646111 TSecr=641801134
0.185521 10.0.12.10 -> 174.137.42.75 HTTP 181 GET / HTTP/1.1
<output omitted>
42 packets dropped
36 packets captured
[-]$
[-]$tshark -i eth1 -R "http.request.uri"
Capturing on eth1
2.932826 10.0.12.10 -> 174.137.42.75 HTTP 181 GET / HTTP/1.1
1 packet captured
[-]$

```

Analyzing captured data

After we have reduced our captured data to a reasonable level, we can now begin with the analysis of the data. Wireshark provides a rich set of easy to use tools. You will find them in the menu under **Analysis** or **Statistics**.

Listing 4. Capture information

```

[-]$capinfos /tmp/out.pcap
File name: /tmp/out.pcap
File type: Wireshark - pcapng
File encapsulation: Ethernet
Packet size limit: file hdr: (not set)
Number of packets: 28234
File size: 29260904 bytes
Data size: 28300663 bytes
Capture duration: 47 seconds
Start time: Fri Oct 5 20:38:03 2012
End time: Fri Oct 5 20:38:50 2012
Data byte rate: 604322.15 bytes/sec
Data bit rate: 4834577.20 bits/sec
Average packet size: 1002.36 bytes
Average packet rate: 602.90 packets/sec
SHA1: 5284fc1b1d17836b0670ec07f751ad38369f49fb
RIPEMD160: 4ffd2e5e6ad5d0577aad6391e77aca5a4d1d2357
MD5: f1fd14e630f7bfffcd8f292545113dd1
Strict time order: True
[-]

```

A good start is to look at the overall capture statistics, you can access them under *Analysis->Statistics*, or command line with the *capinfos* tool (Listing 4). The most important information is about the data rate, round about 5 mbit/s is a good value for my Internet connection, and the average packet size around 1000 bytes per packet is a good value. This was a download of Wireshark from the website, so packets sizing 1500 bytes were travelling to me from the web server, but the acknowledgment to the web server was sent in small packets. The other interesting point is the *Expert Info* where we can find summarized errors, warnings, and other information seen in the capture (Figure 2).

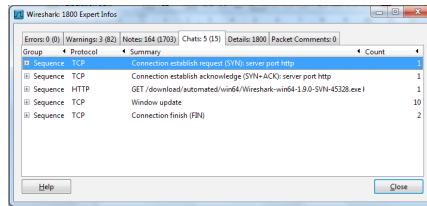


Figure 2. Expert info

Other helpful tools are:

- the IO Graph (*Statistics->IO Graph*) (Figure 3),
- Time Sequence Graph (*Statistics->TCP StreamGraph->Time Sequence Graph* (Stevens),
- or *Statistics->TCP StreamGraph->Time Sequence Graph (tcptrace)*),
- and Round Trip Time Graph (*Statistics->TCP StreamGraph->Round Trip Time Graph*) can help you visualize how your traffic flow is developing over the time. Spikes and holes in the graphs are good indication that something is wrong.

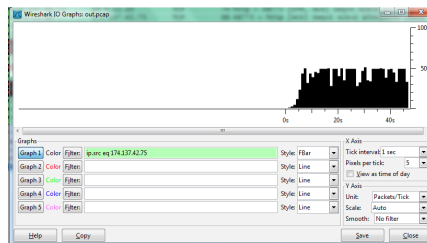


Figure 3. Normal io graph

Security analysis can also be done. You might want to look for unusual traffic like a lot of TCP connect packets or when one host is trying to connect to many hosts, maybe outside of your network. You might also want to search for a specific pattern in your traces, for example, for the Conficker worm you might use `smb.services contains "NetPathCanonicalize"` as filter. This will help you identify the infected hosts.

Exporting data for reporting

Sometimes it is necessary to write a report for a problem or to prepare a presentation, but the graphs are not adequate, or don't fit your presentation style. Wireshark can produce during analysis some graphs, but there is no reporting feature built in. However, you can export the data into several formats, like CSV (*Comma Separated Values*). This is done under *File->Export Packet Dissections->as CSV*, also with *tshark* format the output,

for example, please look at (Listing 5). This data you can process with Office tools like Excel or OpenOffice.

Listing 5. *exporting data as csv*

```
[~]$tshark -r /tmp/out.pcap -T fields -e frame.number -e frame.time_relative -e ip.src
-e ip.dst -e ip.proto -e frame.len -e tcp.analysis.ack_rtt -E header=y -E
separator=, -E quote=d -E occurrence=f
frame.number,frame.time_relative,ip.src,ip.dst,ip.proto,frame.len,tcp.analysis.ack_rtt
"1","0.000000000","10.0.12.10","174.137.42.75","6","74",
"2","0.183815000","174.137.42.75","10.0.12.10","6","74","0.183815000"
"3","0.183845000","10.0.12.10","174.137.42.75","6","66","0.000030000"
"4","0.184419000","10.0.12.10","174.137.42.75","6","241",
"5","0.371743000","174.137.42.75","10.0.12.10","6","66","0.187324000"
```

Where to capture

After we have discussed how we can filter and analyze the data, we should take a look where we can get the data from. Sometimes it is not practicable to capture directly on the client or the server. But it is also possible to add a network tap or use a port mirror on the switch, it is even possible to capture the traffic on the network device and export this in pcap format so that Wireshark can read the capture. Each of this methods has both advantages and disadvantages.

You have seen how to capture data directly on the nodes. To capture data with a network tap or a hub is not more complex, just add it somewhere along the path of the packets. The main disadvantage is that you will have to unplug cables, so this process is disruptive for the traffic and may have other side effects for the connection, for example, most hubs operate with 10mbit speed.

Listing 6. *traffic capture on a Cisco Switch*

```
#configure terminal
(config)#monitor session 1 source interface GigabitEthernet 0/2
(config)#monitor session 1 destination interface GigabitEthernet 0/3
#
```

Port Mirrors on switches are a good idea, as long as you have ports and resources on the switch, because this method is non-disruptive and gives you the possibility to capture a large amount of data. When setting up the wrong mirror port, you might see not the traffic you expect to see or packets will be dropped on the mirror port which are exiting the mirrored port. For example, Cisco Catalyst Switches can mirror traffic, this feature is called SPAN (Switched Port Analyzer), a session would be set up in this way: This will configure the switch to copy all frames from GigabitEthernet 0/2 also to GigabitEthernet 0/3, this will give a system connected to port 2 and

Wireshark installed to trace traffic to and from the system on port 2. Some network devices can capture the data to an internal ring buffer and export this in pcap format, like the Cisco ASA Firewall Series (Listing 7), Cisco Routers (Listing 8) and Juniper Devices. You can use those when you want to capture only a limited amount of traffic, because they have limited availability of memory. If you need more information on how to capture packets on specific hardware, on the websites from the manufacturer, you will find appropriate information.

Listing 7. traffic capture on a Cisco ASA

```
#configure terminal
(config)# ! define interesting traffic
(config)# ! make sure to define both directions
(config)# access-list capture-list permit tcp host 10.0.12.10 host 174.137.42.75
(config)# access-list capture-list permit tcp host 174.137.42.75 host 10.0.12.10
# ! Start the capture
#capture capture-inside interface inside access-list capture-list buffer 100000 packet
    1522
#
#! export the capture
#copy /pcap capture:capture-inside ftp://myhost/mycapture.pcap
```

Listing 8. traffic capture on a Cisco Router

```
#!create the capture access-list
(config)#ip access-list extended capture-list
(config-ext-nacl)# permit ip host 10.0.12.10 host 174.137.42.75
(config-ext-nacl)# permit ip host 174.137.42.75 host 10.0.12.10
(config-ext-nacl)#
#monitor capture buffer capture-buffer size 1024 max-size 1500 circular
#monitor capture buffer capture-buffer filter access-list capture-list
#monitor capture point ip cef capture-point fastEthernet 0 both
#monitor capture point associate capture-point capture-buffer
#monitor capture point start capture-point
#
#sh monitor capture buffer all parameters
Capture buffer capture-buffer (circular buffer)
Buffer Size : 1048576 bytes, Max Element Size : 1500 bytes, Packets : 998
Allow-nth-pak : 0, Duration : 0 (seconds), Max packets : 0, pps : 0
Associated Capture Points:
Name : capture-point, Status : Active
Configuration:
monitor capture buffer capture-buffer size 1024 max-size 1500 circular
monitor capture point associate capture-point capture-buffer
monitor capture buffer capture-buffer filter access-list capture-list
#
#! export capture
#monitor capture buffer capture-buffer export ftp://myhost/cap
#
#! for more options please review the cisco website
```

The shark goes wireless

Capturing wireless control traffic can be done with Wireshark. To capture the control frames, the system must support the monitor mode on the card. Its availability are platform, driver and *libpcap* dependent, on most Linux

systems it is possible to get the card into monitor mode with `iwconfig` or more easy with the `airmon-ng` script, for example, `airmon-ng start wlan0`, on windows, the *AirPcap* adapters from *Riverbed* allows the capture of full raw wireless traffic.

The WLAN traffic summary will look like (Figure 4).

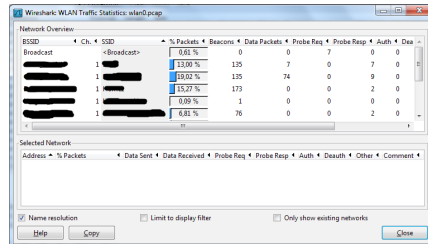


Figure 4. WLAN Traffic summary

Security and Legal Aspects

The use of Wireshark is not without risks. Unauthorized people can come into the ownership of sensitive information, maybe healthcare, bank data, and so on. It is therefore advisable to have a clear policy for the use of Wireshark and other tools. Questions that should be answered are: Who is allowed to capture? How to deal with the captured data? Your policy should also include the need to encrypt the data. If you do not do this, sensible data can leave the company and may have serious legal and financial consequences for the company and you as an individual. In many countries the use of Wireshark and other tools has been banned and placed under strict and heavily regulated laws. Please inform yourself beforehand about the law and consider contacting a lawyer.

Summary

Wireshark is a powerful tool to analyze network data and it can help you improve your network skills.

We have seen that it is pretty easy to capture traffic in the network and that we analyze them for issues. Tracing wireless networks is more demanding, and, when possible, capture the traffic on the wire. In my experience, it is helpful to have a baseline of captures at hand and to update it when there are changes in applications.

On the Web

- <http://www.Wireshark.org> – The Wireshark Homepage
- <http://www.tcpdump.org/> – Home of tcpdump and libpcap
- https://www.cisco.com/en/US/customer/products/hw/switches/ps708/products_tech_note09186a008015c612.shtml – Cisco Catalyst Mirror Ports
- <https://www.cisco.com/en/US/docs/ios-xml/ios/epc/command/epc-cr-m1.html> – Cisco Routers Packet Capture

- <https://supportforums.cisco.com/docs/DOC-1222> – Cisco ASA Packet Capture
- <http://www.aircrack-ng.org/doku.php?id=airmon-ng> – airmon-ng script

Glossary

- SPAN – Switched Port Analyser
- IP – Internet Protocol
- IPv6 – IP Version 4
- TCP – Transmission Control Protocol
- UDP – User Datagram Protocol

Patrick Preuss

Patrick Preuss is working as a network engineer for a large company in Germany. He has more than twelve years of experience in network design and analysis. He can be contacted under patrick.preuss@gmail.com.



Mobile Internet World & Awards 2012

5-7 December 2012| Beijing Marriott Hotel Northeast
The Dawn of a New Mobile Era

Conference Highlights:

500 +
Attendees

80 +
One-on-one meetings

40 +
Speakers

30 +
Operators

20 +
Hours networking

5
Special Awards

OUR VISION:

Mobile Internet World & Awards (MIWA) is the largest and most elite global mobile Internet summit. The novel state of the mobile Internet industry will prove to be a stimulating point of discussion, and will make MIWA one of the most exciting and engaging events, with enterprises from a wide array of industries ranging from services, software, games, end-user device producers, telecom operators, government representatives, industrial organizations, academic elites, investors, media, totally reaching over 250 representatives participating to discuss the developmental changes of the global mobile Internet industry.

Register right now to Save more!



- Organizer: IITA
- Co-organizer: CDME
- Endorser: SVC WIRELESS
- Associate Sponsor: DOLBY
- Presentation Sponsor: one2many
- Co-located with: DP Digital Publishing

Media support

- 上方网 SFW.CN
- yesky.com 天极网
- 移动通信 MOBILE COMMUNICATIONS
- 腾讯互联 TENCENT
- PHONE WORLD
- THE SMART SENSE
- TechWeek europe
- HAKINS IT SECURITY MAGAZINE

AND...YOU CAN' T AFFORD TO MISS!
Specialized and Interactive Workshop-October 31st 2012



Four kinds of the host mode
Marketing : Elim Weng

+86 21 6840 7631
+86 21 6840 7633
<http://www.cdmc.org.cn/mi2012/>
mi@cdmc.org.cn

Wireshark Not Just A Network Administration Tool

Wireshark, a powerful network analysis tool formerly known as Ethereal, captures packets in real time and displays them in human-readable format.

Wireshark was developed by Gerald Combs and is free and open-source. It is used for network troubleshooting, analysis, software and communications protocol development, and education and in certain other ways in hands of a penetration tester as we will learn further in this article. Wireshark is platform independent, and runs on Linux, Mac OS X, BSD, and Solaris, and on Microsoft Windows. There is also a Command Line version called Tshark for those of us who prefer to type.

Where to get Wireshark?

You can download Wireshark for Windows or Mac OS X from [its official website](#). If you're using Linux or another UNIX-like system, you'll probably find Wireshark in its package repositories. For example, if you're using Ubuntu, you'll find Wireshark in the Ubuntu Software Center.

Features of Wireshark

- Distributed under GNU Public License (GPL)
- Can capture live data from a number of types of network, including Ethernet, IEEE 802.11, PPP, and loopback.
- Wireshark can also read from a captured file. See [here](#) for the list of capture formats Wireshark understands.
- Supports tcpdump capture filters.
- Captured network data can be browsed via a GUI, or via the terminal (command line) version of the utility, TShark.
- Captured files can be programmatically edited or converted via command-line switches to the “editcap” program.
- Data display can be refined using a display filter.
- Plug-ins can be created for dissecting new protocols.
- VoIP calls in the captured traffic can be detected. If encoded in a compatible encoding, the media flow can even be played.

- Raw USB traffic can be captured.
- Wireshark can automatically determine the type of file it is reading and can uncompress gzip files

Wireshark Command Line Tools

- tshark – similar to tcpdump, uses dumpcap as packet capture engine.
- dumpcap – network traffic dump tool, capture file format is libpcap format.
- capinfos – command-line utility to print information about binary capture files.
- editcap – remove packets from capture files, convert capture files from one format to another, as well as to print information about capture files.
- mergcap – combines multiple saved capture files into a single output file.
- rawshark – dump and analyse network traffic.

Let us get started – Capturing Packets with Wireshark

After downloading and installing Wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface (Figure 1).

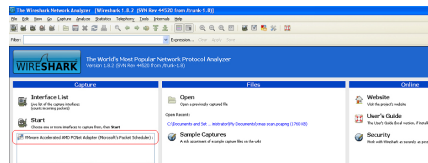


Figure 1. Packet Capture

Or you can go to the menu bar and click on Capture > Interfaces and select the interface on which you want to capture the traffic (Figure 2).

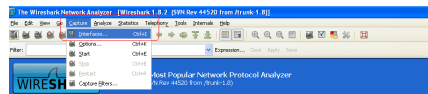


Figure 2. Packet Capture

Here we click on the Vmware network adaptor and start capturing the packets (Figure 3).

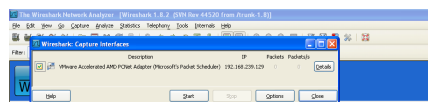


Figure 3. Packet Capture

Let us try some basic packet capture. Let us browse to www.google.com and see the traffic generated.

The local computer 192.168.239.129 queries the DNS server 192.168.239.2 to find out who is google.com. The DNS query response by 192.168.239.2 is displayed which gives the IP addresses of multiple google web servers. This is followed by the three way TCP handshake (SYN, SYN-ACK, ACK) with one of the google web server on 74.125.236.183 as shown Figure 4.

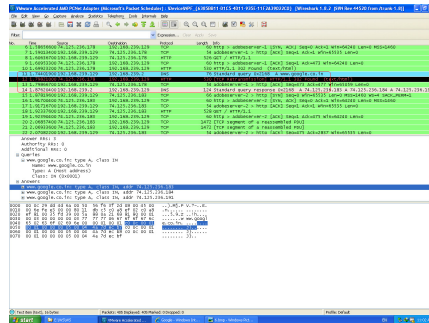


Figure 4. Google Browsing Traffic

The HTTP traffic which commences post TCP handshake commences with a GET request as shown. Here we can use another feature of Wireshark to follow this particular HTTP traffic. For this, we right click on the GET request and select *Follow TCP Stream* (Figure 5).

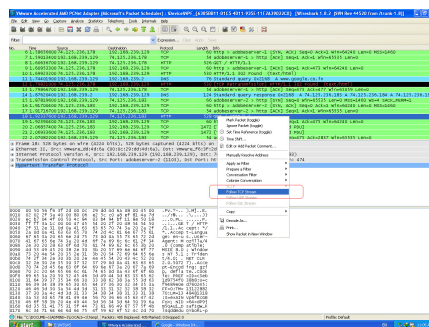


Figure 5. Follow TCP Stream

We can view the entire HTTP transaction in a new window (Figure 6).

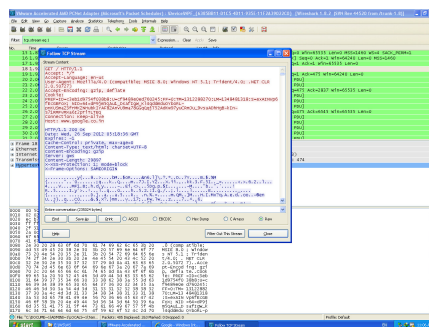


Figure 6. HTTP Traffic Stream

Separating out Network Traffic of our interest – Use of Display Filters

Wireshark provides an interesting feature of filtering the network traffic using display filters. Let us look at some of these filters and how we can mix and match them to get down to an item of our interest.

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type “dns” and you’ll see only DNS packets. When you start typing, Wireshark will help you auto complete your filter. Another way to achieve the same result is to go to the *Analyse* tab in the main menu bar and select display filter.

Let us say we want to check out all DNS packets which are from Authoritative DNS Servers. After typing DNS, we can scroll down the drop down list and select *dns.flags.authoritative* (Figure 7).

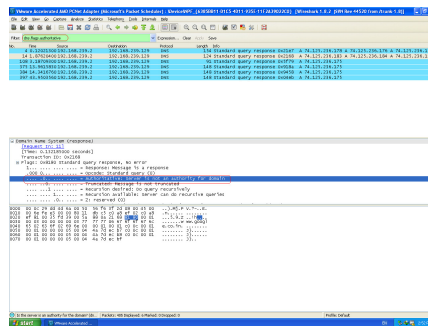


Figure 7. DNS Authoritative Flag

The selected DNS packet shows that the DNS server is not an authoritative server for the requested domain as the Authoritative Flag is not set.

Playing Around with Filters Using Operators

Some basic operators we can use with display filters are as shown.

- Equal: eq, =
- Not Equal: ne, !=
- Greater than: gt, >
- Less Than: lt, <
- Greater than or equal to: ge, >=
- Less than or equal to: le, <=

Example

Say we want to see all HTTP GET requests in the captured traffic. We can type `http.request.method == "GET"` into the Display Filter box and get all the GET requests made by the user (Figure 8).

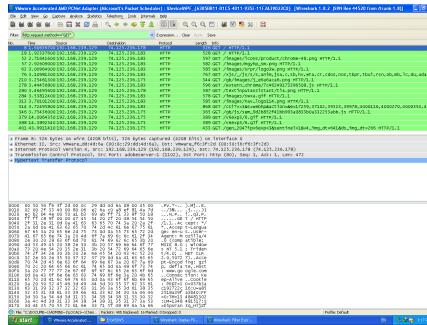


Figure 8. HTTP GET

Over with Basics, Time to Have Some fun now..

Let us now see if we can sniff unencrypted passwords. So, I need to find an insecure website which uses http for sending login credentials instead of https. Unfortunately, this fun is almost over now as most of the websites have shifted to https. This is a test website for checking web application vulnerabilities (<http://demo.testfire.net>) (Figure 9).

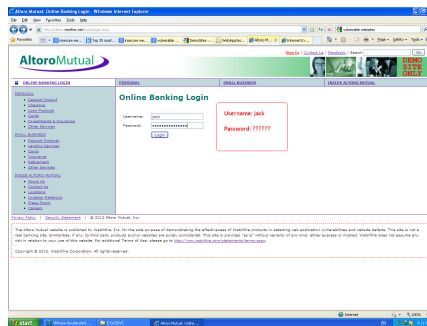


Figure 9. Sniff Password

So, let us use the filter feature in Wireshark to just only filter the HTTP POST method. Type – `http.request.method == "POST"` into the display filter box and let us see what we get. Two packets with HTTP POST request are filtered out, we select the packet of our interest and view packet details in the lowermost window. I think we just got lucky here.. (Figure 10).

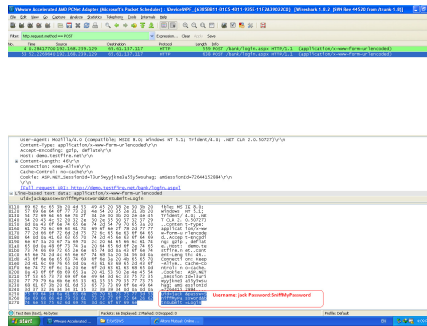


Figure 10. Sniff Password

How can Wireshark Help me in Network Security?

Wireshark can give a network administrator a very good idea of what is happening on his network. Although not an Intrusion detection tool, it can easily help in checking some security policy violations.

Identifying Bittorrent Downloads

The protocol used for peer to peer transfers is the giveaway here. We can view only the BitTorrent packets by typing *bittorrent* in the filter box. You can do the same for other types of peer-to-peer traffic that may be present, such as Gnutella, eDonkey, or Souseek (Figure 11).

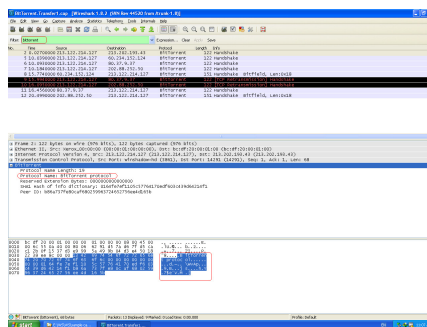


Figure 11. Identify Bittorrent

We can also view the network usage based upon protocol by going to Statistics tab on Menu bar and selecting Protocol hierarchy. Here we see that the bittorrent traffic is occupying almost 70 % of overall network traffic. So much for downloading movies at the wrong time and place (Figure 12).

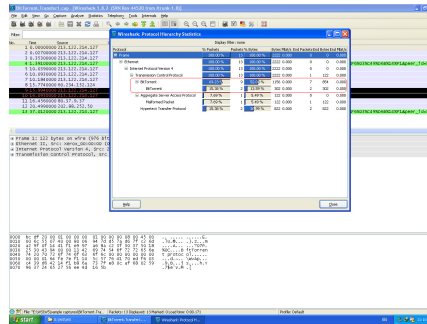


Figure 12. BitTorrent Stats

Identifying Facebook Usage

Can't live with or without it? Well, your network admin may be watching if your organisation does not allow it.

Sites like Facebook often use several servers to provide content to users. We can't just filter one ip address and be done with it. It can involve many different addresses, and usually changes per user. The simplest way to set a filter for Facebook users is to use the "tcp contains facebook" filter (Figure 13).

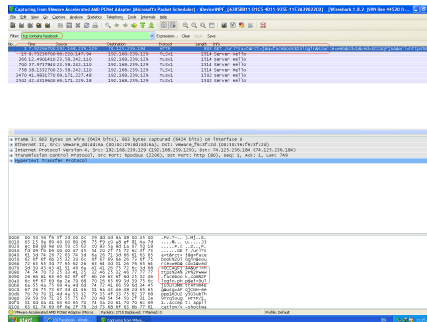


Figure 13. Facebook

So once, we are done with the so called bad guys on the inside of our network, let us watch out for the bad guys outside the network. Well, having said that these attacks can be better done from inside the network bypassing all our perimeter security and taking advantage of the trust placed by the organisation on its employees.

Identifying Port Scans

Let us now see how a TCP SYN scan would appear on Wireshark interface. TCP SYN scan is also known as half open scan because a full TCP connection is never established. It is used to determine which ports are open and listening on target device.

We can see that the attacker IP 192.168.239.130 is ending packets to victim IP 192.168.239.129 with the SYN Flag set (Figure 14).

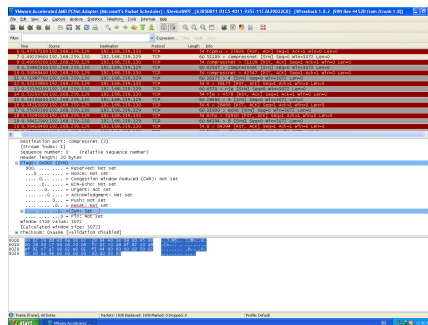


Figure 14. SYNscan

The victim IP responds with a RST ACK packet. This indicates that the port is closed.

In case if SYN /ACK is received, it indicates that the port is open and listening

X-Mas Scan

The X-Mas scan determines which ports are open by sending packets with invalid flag settings to target device. This scan is considered stealthier than SYN scan as it may be able to bypass some firewalls and IDSes more easily.

The attacker send TCP packets with FIN, URG and PSH flags set and gets RST ACK reply back. This indicates that the port is closed. An open port will simply drop the packet and not respond.

X-Mas scan would appear like this on Wireshark (Figure 15).

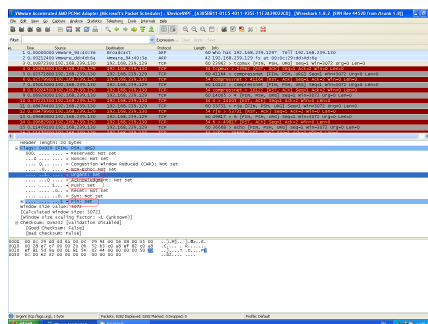


Figure 15. XmasScan

Identifying Malware Infection

So someone has already clicked, despite all the security training, presentations, workshops, etc, etc. In fact, we are slowly reconciling to the fact that no matter what you do, the user will always fall to the ever tricky

ways of attacker and this should be the basis of our risk assessment. If we can save our networks and data even after a machine has got compromised, we have a chance to survive in this world of zero days.

Wireshark can help us in identifying malware infections on our network. Most of the modern malware operate in a client server mode and allows the attacker to have full remote control of the target machine.

Let us consider a case scenario wherein an employee indulges in indiscreet surfing on internet. As is likely, the malicious websites visited by the employee would try to download malicious code on the employee computer (you can find nothing for free in life and certainly not on internet). If we have a packet capture of the network traffic, it can be analysed by using Wireshark. Let us see how it happens. For this, we go the File menu and select *Export Objects > HTTP* (Figure 16).

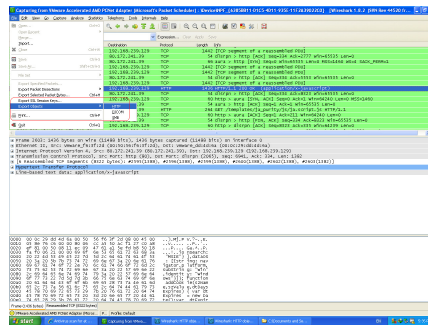


Figure 16. Export Objects

Wireshark provides us with a list of all HTTP objects downloaded on the employee machine. Here we select a file “*javascript.js*” and save it to a desired location on the local computer (Figure 17).

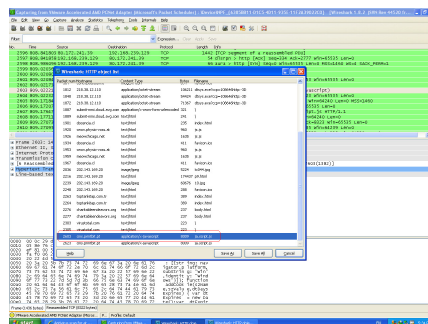


Figure 17. Jssaveas

Our suspicion about this file is confirmed as the antivirus alert pops up immediately on our desktop indicating that the file is malicious (Figure 18).

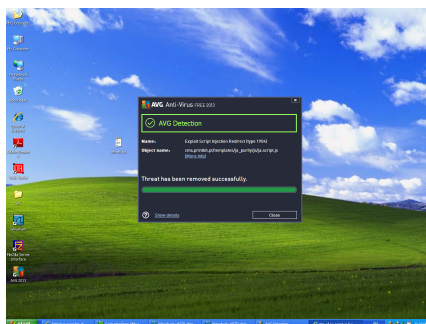


Figure 18. Jsddetection

So, now we are level zero of Wireshark proficiency. To dig deeper (and I'm sure it is worth it), we have the option of attending free live training webinars by Laura Chappell, or go through her Wireshark Network Analysis guide and get ourselves certified as Wireshark Certified Network Analyst.

Arun Chauchan

Joint Director CIRT Navy at Indian Navy

Wireshark: The Network Packet Hacker or Analyzer

The purpose of this article is to provide the overview of the powerful tool Wireshark. The document also explains how to build a working setup to analyze Ethernet standardized network packets.

In order to run wireshark, there are following pre-requisites that must be present.

- Linux/Windows desktop host machine.
- Host machine must have Ethernet interface.
- The user should have basic Linux/Windows environment knowledge.
- PC should be connected to network via a Ethernet cable.

Table 1. Acronyms and Abbreviations

Wireshark	Wireshark is an open source network packet sniffer tool
IP	Internet Protocol
GSM	Mobile phone communication network terminology (Global System for Mobile Communications)
VoIP	Voice over IP

Overview

Wireshark is an open source tool for capturing and analysing network packets, from standard network protocols such as Ethernet, TCP, UDP, HTTP to GSM Protocols like LAPD. Wireshark works like a network

packet X-Ray and can listen to network traffic to help identify problems related to protocols, applications, links, processing time, latency and more. This tool expands packet header and data information which is user friendly understandable information for debugging networking issues.

On running the Wireshark Analyser tool, network packets are displayed in the *Graphical User Interface* (GUI) at run time. Each packet shown in GUI can be expanded to view various header fields of the network packet.

Wireshark supports IPv4, IPv6, 6lowPAN and many more networking standards & protocols.

Wireshark tool usage

- Debugging Internet Protocol TCP and UDP which are the most commonly used protocols for communication. Debugging for the following problems when analysing TCP-based applications using Wireshark
 - Zero Window
 - Window is Full
 - Keep-Alive
 - Window Update
 - Previous Segment Lost
 - Retransmissions/Fast Retransmissions
 - Duplicate ACKs
- Wireshark is a useful tool to determine the cause of slow network connections.
- To expose problems for VoIP using Wireshark.
- To expose LAPD/ABIS GSM protocol message debugging for missing acks session close etc.

Wireshark is an open source tool which can be extended for any communication protocols message debugging.

How to setup Wireshark

Connect Wireshark host machine to a hub to capture network packet flow (Figure 1).

Configuring setup on Windows and Linux system:

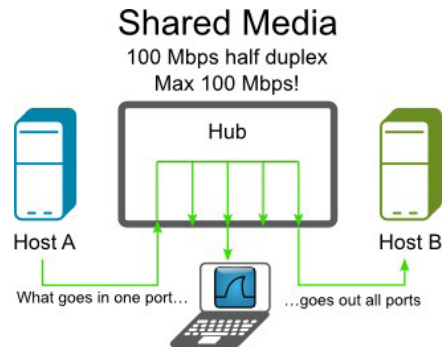


Figure 1. Setup block diagram

The following steps show you how to configure Wireshark:

- Install Wireshark: On Windows, download [Wireshark](#) and install with the default selections, including WinPcap. On Linux, enter the commands with root privileges:
 - yum search wireshark
 - yum install wireshark
 - yum install wireshark-gnome
- Configure the interface to be analysed
 - Start Wireshark.
 - Select the “Capture | Interfaces” menu item.
 - Choose the network interface exhibiting issues and click Start.
- Launch the application you want to analyse (the TCP client, for example).
- To configure a filter with a focus on Perforce network traffic click the Expression item next to the Filter item.
- Select the Capture | Stop menu item when you have completed reproducing the issue.
- To save the results, select the File | Save as... menu item to save the output as a .pcap file. This file can be sent to Perforce for analysis.

Linux based wireshark setup block diagram (Figure 2).

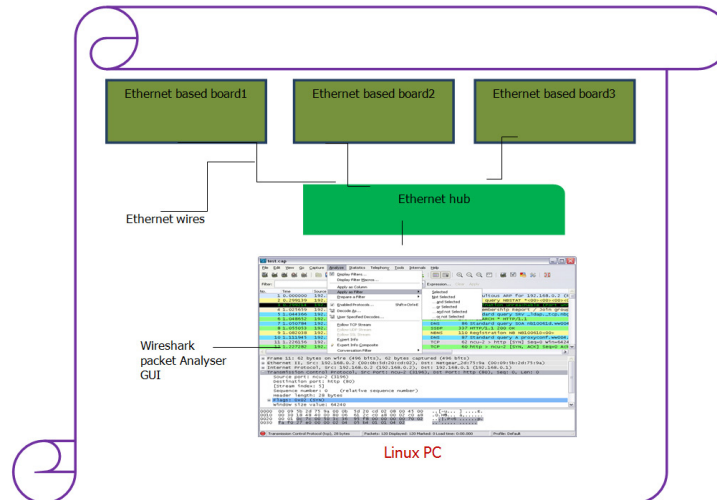


Figure 2. Setup Linux PC

How wireshark works (Technical block diagram)

It taps the packet from wire and a handler is called for packet parsing and display. As show Figure 3.

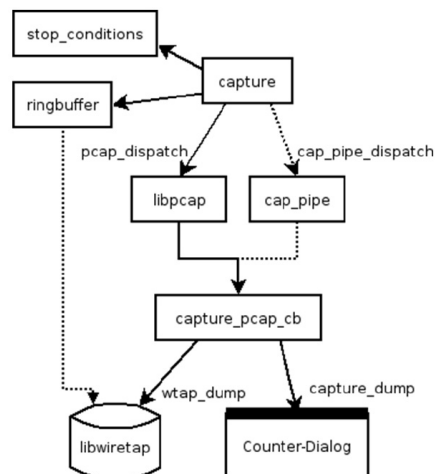


Figure 3. Wireshark packet tapping and parsing

Wireshark Packet Analyser Screenshots

- The Figure 4 displays the Wireshark main window with packets captured from the network

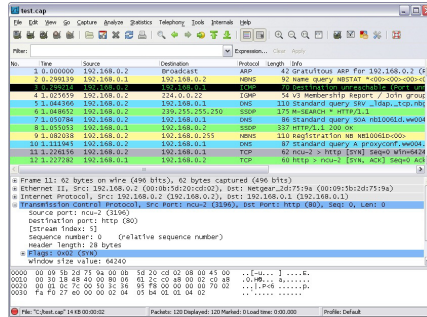


Figure 4. Wireshark packet capture main window

- Wireshark statistics view window (Figure 5)

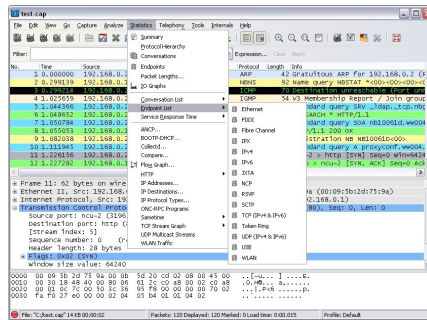


Figure 5. Wireshark statistics view

- Wireshark time reference window (Figure 6).

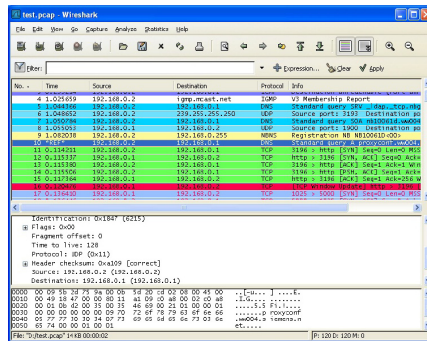


Figure 6. Wireshark time reference window

- Wireshark packet analyse view (Figure 7).

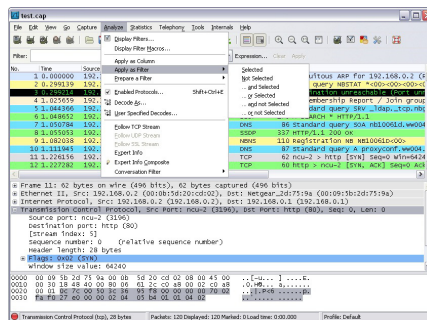


Figure 7. Wireshark packet analyser view

Conclusion

Tapping into the communications in a passive manner enables you to identify communication problems. Mastering analysis of communication protocols is critical when identifying the source of those problems and differentiates. Wireshark shows each bit and byte of the filtered protocol packet along with sensible header byte information to show detailed information that aids in problem solving within the network. Network analysis is one of the key skill sets all IT and security professionals should master. Wireshark assists network professionals to learn how the protocols and applications interact with each other.

Anand Singh



IT Security Courses and Trainings

IMF Academy is specialised in providing business information by means of distance learning courses and trainings. Below you find an overview of our IT security courses and trainings.

Certified ISO27005 Risk Manager

Learn the Best Practices in Information Security Risk Management with ISO 27005 and become Certified ISO 27005 Risk Manager with this 3-day training!

CompTIA Cloud Essentials Professional

This 2-day Cloud Computing in-company training will qualify you for the vendor-neutral international CompTIA Cloud Essentials Professional (CEP) certificate.

Cloud Security (CCSK)

2-day training preparing you for the Certificate of Cloud Security Knowledge (CCSK), the industry's first vendor-independent cloud security certification from the Cloud Security Alliance (CSA).

e-Security

Learn in 9 lessons how to create and implement a best-practice e-security policy!



Information Security Management

Improve every aspect of your information security!

SABSA Foundation

The 5-day SABSA Foundation training provides a thorough coverage of the knowledge required for the SABSA Foundation level certificate.

SABSA Advanced

The SABSA Advanced trainings will qualify you for the SABSA Practitioner certificate in Risk Assurance & Governance, Service Excellence and/or Architectural Design. You will be awarded with the title SABSA Chartered Practitioner (SCP).

TOGAF 9 and ArchiMate Foundation

After completing this absolutely unique distance learning course and passing the necessary exams, you will receive the TOGAF 9 Foundation (Level 1) and ArchiMate Foundation certificate.

For more information or to request the brochure please visit our website:

<http://www.imfacademy.com/partner/hakin9>

IMF Academy

info@imfacademy.com

Tel: +31 (0)40 246 02 20

Fax: +31 (0)40 246 00 17



WiFi Combat Zone: Wireshark versus the neighbors

If you're one of the regular readers of Hakin9, then you know that there are several means by which your neighbors could have penetrated your WiFi LAN. Do you ever wonder if it's already happened? Would you like to learn how to monitor anybody that's abusing your network?

Then take a look at “WiFi Combat Zone: Wireshark versus the neighbors”, where we will take a deep look at the well-known, free “Wireshark” Ethernet diagnostic software, concentrating on its use while monitoring the activities of uninvited guests on our networks.

If you're one of the regular readers of Hakin9, then you know that there are several means by which your neighbors could have penetrated your WiFi LAN. Do you ever wonder if it's already happened? Would you like to learn how to monitor anybody that's abusing your network?

You've come to the right place!

In today's message, we will take a deep look at the well-known, free “Wireshark” Ethernet diagnostic software, concentrating on its use while monitoring the activities of uninvited guests on our networks.

Wireshark has been around for a long time! I first stumbled upon it back in the late 1990s, when it was known as “Ethereal”, the product of a talented American network engineer named Gerald Combs. I was thrilled with it. At the time, I was designing a new, commercial network security system for my own small company, and I had been trying to persuade investors that the future would bring increasing need for security products. Using Wireshark with their permission, I was able to capture usernames and passwords on the Ethernet LANs of potential investors. They had all heard that this sort of thing was possible, but prior to the appearance of Ethereal, the necessary tools had been very expensive.

When I told them that Ethereal was free, legal, easy to use, and compatible with almost every inexpensive PC then in existence, my investors got out their checkbooks! I've been using it ever since.

Wireshark Architectures

Wireshark software is easy to install, and the installation process follows the general and well-established norms for each computing platform. It will run on almost any personal computer, using LINUX, MAC OS-X, Windows, and several of the most popular versions of Unix. Free versions for Windows and Macintosh platforms can be downloaded from www.wireshark.org. Even the source code is available there, for public examination. Linux users could install from the source code, but most Linux distributions include Wireshark as a precompiled application within their “repository” libraries, according to the common new Linux traditions.

But there is a problem....

Although it is easy to obtain and install Wireshark, it is generally NOT easy to get it to intercept WiFi traffic in a broad, general-purpose way.

Interception and examination of WiFi traffic with Wireshark is NOT the same as using the well-known “Promiscuous Mode” to examine conventional Ethernet traffic.

Although all WiFi adapters are capable of gathering WiFi signals from every compatible 802.11 emitter within range, the “driver” software that connects your hardware WiFi adapter with your operating system will discard any of those signals that are directed toward other computers unless it has been specifically designed to support what WiFi engineers call “Monitor Mode”. And here’s the problem: Most popular, low-cost WiFi drivers do NOT support Monitor Mode (This is especially true of drivers written for the Microsoft Windows operating system).

Unless you are among the fortunate few with a WiFi card whose device driver software supports Monitor Mode, your copy of Wireshark will display only packets directed at your own computer, and “broadcast packets” that are deemed to be safe when broadcast to everybody on your LAN. You won’t be able to see conversations between the other computers and nodes of your network, and you won’t be able to monitor the details of the traffic they exchange on the Internet.

For the remainder of this article, we are going to assume that you suffer from these constraints like most people.

Don’t despair.... We have two simple, low-cost solutions for you! You WILL be able to monitor your neighbors (and others) using WiFi to connect

to your LAN as they send and receive information through your Internet connection. We call these solutions “Wireshark Intercept Architectures”. They will require you to make some changes to your home or small office LAN, but the changes are simple and very low in cost. As illustrated in the two figures below, the two architectures are: Figure 1 and Figure 2.

As shown in Figure 1 and 2, an Ethernet Hub is central to all of our plans. An Ethernet Hub looks a lot like a common “Ethernet Switch”, and although it connects into your network in the same way, it is NOT the same thing. When you go shopping for an Ethernet Hub, you’ll be looking for a low-cost, profoundly dumb device.

Although Ethernet Switches use more modern technology and are more common, Ethernet Hubs are still readily available. The difference between an Ethernet Hub and an Ethernet Switch is fundamental to our interception architectures. Here are the definitions: Figure 3.

Ethernet Hub: An electronic device that expands the number of Ethernet connections by a process of mindless signal replication, so that any Ethernet signal that enters into the hub through any of its connectors is replicated at all of the others (Figure 4).

Ethernet Switch: An electronic device that expands the number of Ethernet connections by a process of intelligent signal switching. The source address of every Ethernet frame entering the switch through any of its connectors is examined and recorded in a table, associating it with the connector through which it arrived, so that the switch learns the Ethernet addresses of equipment attached to each connector. The destination address of every Ethernet frame entering the switch through any of its connectors is also examined and compared with the table. If the switch does not yet know which connector leads to the addressed destination, then the switch behaves exactly like an Ethernet Hub, “broadcasting” the packet to every connector to maximize the likelihood of proper transmission. On the other hand, if the switch already knows the proper connector for delivery, it sends the packet ONLY out that connector to minimize traffic congestion (Figure 5).

By now it should be clear why we want to insert an Ethernet Hub into our network: It creates a perfect “wiretap” for Wireshark! Wherever you insert your Ethernet Hub, you can connect an additional computer, running Wireshark, and you can then see ALL of the Ethernet traffic traversing the Hub. It doesn’t matter whether the traffic originated on an encrypted WiFi

link, or through hardwired Ethernet: you get it ALL, and the computer hosting Wireshark won't even need a WiFi adapter! (On the other hand, an Ethernet Switch in the same position would filter out all of the most interesting traffic, sending only Ethernet traffic that is designated for broadcast to everybody).

Take a look at Figure 1. In this architecture, we assume that the WiFi Router at your network's "head end" is separate from your broadband modem. (About half of the world's domestic WiFi networks look like this.) Before beginning this exercise, a single Ethernet cable led between the Broadband Modem and the WiFi Router's "Internet" connector. The Ethernet Hub that we've inserted between the Broadband Modem and the WiFi Router allows the Wireshark Host to see ALL of the Internet traffic for every user of the network.

Now Take a look at Figure 2. In this architecture, we assume that your WiFi Router (designated "WiFi Router 1") has a built-in broadband modem, so you can't get access to an Ethernet segment upstream of your WiFi traffic. This is another very common situation, because most domestic Internet Service Providers install an "all in one" WiFi Router and Broadband Modem combination. In this situation, we chose to install a second WiFi Router, designated "Honeypot" router in the illustration. An Ethernet Hub and Wireshark host are then connected between the 2 routers, more-or-less duplicating the wiretap situation shown in Figure 1.

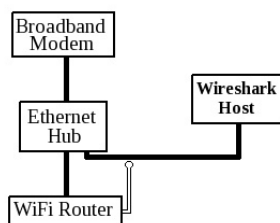


Figure 1. Ethernet Hub between WiFi Router and Broadband Modem

Obviously, the architecture of Figure 2 allows our Wireshark host to see all of the Internet traffic exchanged through the Honeypot Router, but it cannot see Internet traffic exchanged through the original WiFi Router.

Accordingly, we must force any unauthorized users to switch to the Honeypot Router.

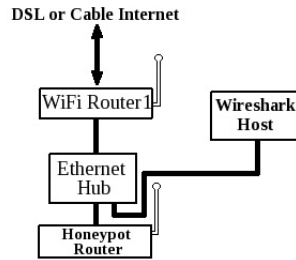


Figure 2. Honeypot WiFi Router and Ethernet Hub

How do we do that? Easy! We just change the WPA encrypting key of WiFi Router 1, and we leave the “Honeypot Router” running WiFi in the clear, without any encryption. All of the users will immediately face a decision: They can ask us for the new WPA key for their familiar WiFi Router 1, or they can experiment with the Honeypot Router’s access.



Figure 3. Ethernet Hub

As you have no doubt surmised, all of the “Interesting” traffic will go for the Honeypot router, and you’ll be able to monitor it!



Figure 4. Ethernet Switch

The Wireshark software

Once Wireshark is installed on your computer, you can begin capturing traffic. You will need to designate a network “Interface” whose traffic you want to monitor. Most computers nowadays have more than one Ethernet interface (Usually a hard-wired Ethernet connector and a WiFi card), and Wireshark’s administrative interface displays a prominent “Capture”

Section where you can activate a “live” list of available interfaces. Each interface in that list is accompanied by a counter that continuously displays the number of Ethernet packets that have been observed.

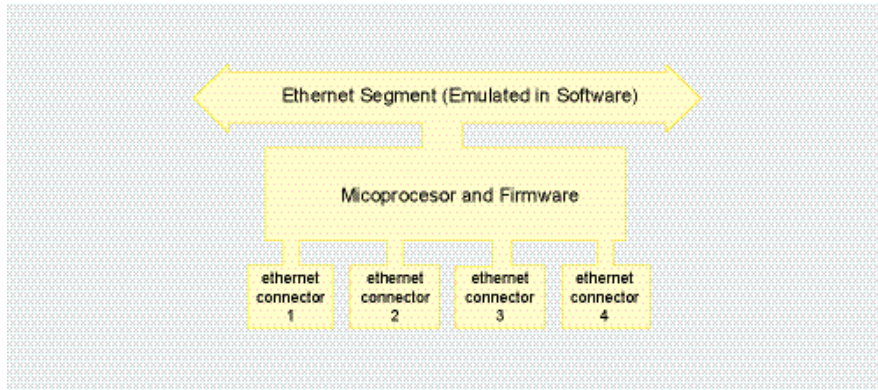


Figure 5. Ethernet Switch Internals. An Ethernet Switch is a lot like an Ethernet Hub, but it includes microprocessor-based intelligence so it can avoid broadcasting most Ethernet signals. Instead, it learns the specific and appropriate destination for each Ethernet frame it processes, and forwards each incoming message fragment only to the appropriate Ethernet connector. This can increase network efficiency and privacy, but it interferes with our desire to monitor all network traffic. For our purposes in this discussion, a Hub is better!

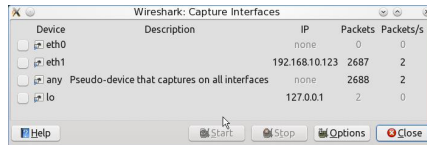


Figure 6. Wireshark's "Capture Interface" Selector

Figure 6 illustrates this list after 2,687 packets had been observed through interface “eth1” (If you just want to examine all packets from all interfaces, you can select the interface labeled “any”).

Once you choose an interface and press the prominent “Start” button, your display will look a lot like Figure 7.

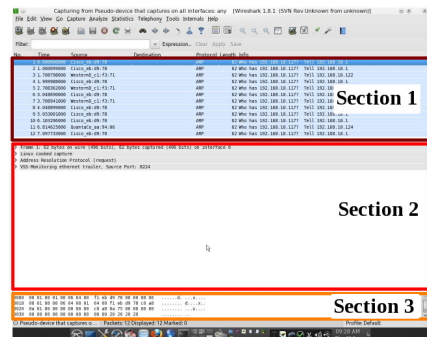


Figure 7. Wireshark in action, showing 3 main sections tiled beneath the usual set of dropdown menus

Beneath the usual arrangement of drop-down menus and icons, your display will be dominated by three large sections tiled on top of one another, each of which will span your entire display window from left to right. You can

re-size each of these 3 areas by left-clicking and dragging on the dividing horizontal boundaries between them.

From top to bottom, these three sections are:

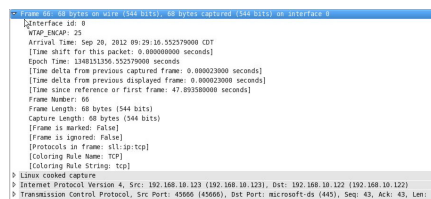
Section 1 of 3

A scrolling list summarizing all captured frames. Each frame is described on a separate horizontal row, identified by a sequence number and its arrival time. Additional fields reveal the frame's source address, destination address, protocol type, and a brief explanation. You can use your mouse to highlight one of the lines in this area for further exploration. In Figure 7 we have highlighted Packet #1, which is identified as an "ARP" frame from Ethernet Address "Cisco_eb:d9:78".

Section 2 of 3

A Protocol Interpretation Area revealing additional information about the Ethernet frame highlighted in the scrolling list. Because Ethernet frames can contain many different types of data packets, Wireshark has been designed to use this area dynamically, and with deep intelligence. Although the general format and arrangement of this area will remain constant, the details change as appropriate to help you explore different kinds of Ethernet frames and as you "drill down" into their contents. As shown in Figure 7, this area is dominated by a series of horizontal lines, each commencing with an "arrowhead" icon to indicate the presence of additional details that can be accessed with a mouse-click.

This arrangement mimics the general organization of Ethernet frames, which can contain packets within packets within packets, and each of those inner packets consists of several "fields" whose purpose and format have been standardized by committees of engineers (who had to come to agreement before data could be interchanged).



```
Frame 60: 68 bytes on wire (544 bits) (68 bytes captured (544 bytes on interface 0)
on interface 0
Ethernet II, Src: VMware, Prio: 25
Arrival Time: Sep 26, 2012 09:29:10.552379000 CDT
[Time shift for this packet: 0.000000000 seconds]
Epoch Time: 134811356.552379000 seconds
[Time delta from previous captured frame: 0.000023000 seconds]
[Time delta from previous displayed frame: 0.000023000 seconds]
[Time since reference or first frame: 47.893580000 seconds]
Frame Number: 60
Frame Length: 68 bytes (544 bits)
Capture Length: 68 bytes (544 bits)
Frame is marked: False
Frame is ignored: False
Protocols in frame: eth II, ip, tcp
[Coloring Rule Name: TCP]
[Coloring Rule Setting: top]
3 [Linux cooked capture]
4 Internet Protocol Version 4, Src: 192.168.10.123 (192.168.10.123), Dst: 192.168.10.122 (192.168.10.122)
5 Transmission Control Protocol, Src Port: 8080 (8080), Dst Port: microsoft-06 (445), Seq: 65, Ack: 93, Len: 0
```

Figure 8. Any of the lines in Section 2 can be expanded for further detail by left-clicking on its arrowhead icon. Here we see the first line expanded, revealing details about the entire, selected Ethernet frame. Note that there are 3 additional lines beneath that first one, each representing content that is buried correspondingly "deeper" within the frame, and that each of those 3 additional lines has its own arrowhead icon, indicating the presence of additional, available details that can be accessed with a simple click of the mouse

Thus the top line in Area 2 of Figure 7 summarizes the entire, corresponding Ethernet frame at the “highest” level. Additional lines beneath that one focus on embedded packets or significant field areas within the frame, with “deeper” embedded frames corresponding with lines beneath upper ones. Clicking on the arrowhead icon at the left of any of these lines will invoke additional, expert logic to analyze the contents of the corresponding data, revealing its structure and purpose in the vocabulary of the engineers who designed and standardized it.

Take a look at Figure 8, showing the way Area 2 examines the 66th captured Ethernet Frame, after left-clicking on the arrowhead icon to expand the very first horizontal line. As you can see, the contents of that summary line have been GREATLY expanded to reveal more information about the entire packet.

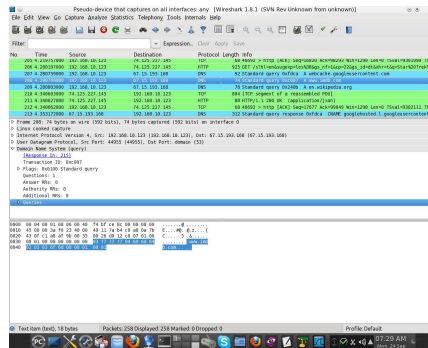


Figure 9. Wireshark’s examination of a more interesting Ethernet frame containing a Domain Name System query packet from a computer operating within our own local IP subnet. Note the text at the bottom identifying the “Internet Movie Database” www.imdb.com. It looks like somebody is going to be looking for movie entertainment....

Section 3 of 3

Return to Figure 7, where you can see Section 3 across the bottom. In this area, Wireshark displays all of the “raw” data within the selected Ethernet frame, without trying to analyze its structure. The data is “dumped” in Hexadecimal across the left side of Section 3, revealing the relative position and precise value of each data byte. If you are comfortable with Hexadecimal math, you can get to “bedrock” using this data dump, even if you encounter an Ethernet frame using a protocol that is completely undocumented. The right side of Section 3 tries to show additional insight, on the assumption that some of the characters may be formatted according to the popular conventions of the “ASCII” character set. Thus, if the data contains a printable word or phrase formatted in the usual way, you’ll see it

here (It is commonplace to see usernames and passwords in this area when unsophisticated, non-encrypted protocols are in use).

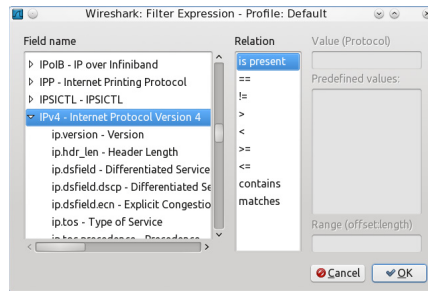


Figure 10. Wireshark's "Filters" tool allows you to filter unwanted information from view. In this example, we are preparing to hide all frames that do NOT contain an IPv4 packet

Capture Everything!

After you begin capturing Ethernet data as described above, you'll notice that the list of data in Section 1 will scroll up as additional frames appear at the bottom. Within a few minutes you'll probably capture thousands of frames, and you may want to stop capturing.

Click the "Capture" drop-down menu heading at the top of your display, and then select "Stop". No further data will be captured, and the scrolling list will stop moving, giving you time to explore individual frames already captured.

At this point you can use the "Save As" option from the usual "File" drop-down menu to save a copy of the captured packets. I recommend that you take this step whenever you've captured traffic that you suspect may contain anything interesting (This is a reversible process; you can load the saved file for further analysis whenever you need to).

Explore the Details

Click on one of the horizontal lines in Section 1, and you'll see associated details in Sections 2 and 3. Click on the resulting, little "arrowhead" icons in Section 2 and you will see further details and labels identifying the purpose and structure of the selected areas. Sometimes, as you explore areas of Section 2, you may notice that areas of the data in Section 3 change color to help you identify the raw data that's associated with the area under examination.

Real expertise with Wireshark will come as you select an individual frame in Section 1 and then use Section 2 to explore its contents, referring to Section 3 as appropriate to read any text messages that it may contain.

All of this will take time! As you will observe, there are a great many different kinds of data packets that can be wrapped up inside Ethernet frames. Most of these won't be very interesting. The great preponderance of Internet traffic is mundane stuff. But every once in a while, you'll find a gem! Pay special attention to the "Source" field in Section 1. Watch for IP addresses from your own local subnet, paying special attention to any that are unfamiliar or that you have not specifically authorized as part of your own network. (Usually these local IP addresses will begin with "192.168", and the subsequent address digits will be assigned by your router according to guidelines you've set up through its management menus.) If neighbors or other unauthorized people are using your network, their packets will be among this group.

For example, take a look at Figure 9, in which we examine frame #208, originating from IP address 192.168.10.123. Obviously this IP address comes from our own, local subnet, so it's likely from a computer that's very close by. From Section 1 we can see that it's a DNS packet. Section 2 reveals further that it's a Domain Name System query. By clicking on the associated arrowhead icon in Section 2, we can force Section 3 to highlight the associated data, where we can see that somebody is requesting the IP address of the well-known "Internet Movie Database" at www.imdb.com. This is EXACTLY the kind of behavior that we might expect from an unsophisticated neighbor casually using our Internet connection via WiFi. At this point, it might be wise to browse into the management interface of our WiFi router to see when IP address 192.168.10.123 was issued, and the hardware address of the Ethernet adapter it uses....

More Wireshark tools: "Analyze"

Wireshark's dropdown menus offer additional tools that you might enjoy. For example, after selecting a line representing TCP traffic in Section 1, take a look at the "Analyze" dropdown menu. An option to "Follow TCP Stream" is prominent. Click that option and you'll see a very interesting summary of that TCP packet and all of the other TCP packets comprising the associated TCP session, which could span a long period of time. All of those TCP packets will be located from your captured data, sequenced into proper order, and formatted for your convenient viewing. If this TCP Stream is like most, it will contain printable words and phrases that will be

prominently displayed. This is one of the best ways to get a quick, high-level understanding of the messages traversing your network (Similar analysis tools are also available for examination of sequenced UDP and other session-oriented traffic).

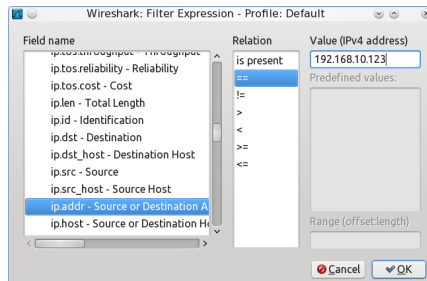


Figure 11. Sometimes additional information is needed in order to complete construction of an appropriate Wireshark display filter. In this case, the filter will exclude all frames unless they are communicating with IP address 192.168.10.123

More Wireshark tools: “Filters”

After capturing thousands of Ethernet frames, you will want to sort through them quickly and easily. For example, you may want to concentrate only on those originating from or going to IP address 192.168.10.123. You can easily use the “Filter” facility to eliminate all other frames from the display list. This is done by clicking on the prominent “Expression” button (as shown near the top of Figure 9), near the blank “Filter” box).

A long, scrollable list of “Field Names” will appear. Scroll that list down to “IPV4” and then click the associated arrowhead icon for further expansion, as shown in Figure 10. Now scroll down further, among the newly displayed ip subfields, to select “ip.addr”. Then, as shown in Figure 11, click within the “Relation” box to select “==”. Finally, type the target IP address “192.168.10.123” into the “Value” box. This will automatically construct what Wireshark calls a “Display Filter” meeting our requirements. From that moment onward, only captured frames originating from or sent to IP address 192.168.10.123 will be displayed, allowing us to concentrate our efforts on the most interesting traffic for our chosen situation.

Conclusions

Wireshark is a very powerful, free software tool that will allow you to examine every detail of traffic on your Local Area Network, including a great many things that casual users assume they can keep private. By configuring your network with an Ethernet Hub near your main Internet

connection, you will be able to connect Wireshark strategically so that you can see the contents of WiFi (and other) traffic exchanged on the Internet. If somebody is abusing your network, you will be able to monitor their activities whenever they happen to use a routine, unencrypted protocol for Internet access.

This will require patient research, because the vast majority of the Ethernet frames that you capture will contain traffic that is either uninteresting, too complex to allow easy analysis, or has been encrypted. However, even the most clever users will eventually access resources that can easily be examined, and by studying their activities with Wireshark, you will be able to determine the IP addresses that they use on your network, the amount of time they spend connected, the amount of traffic they generate, the probable manufacturer and Ethernet address of their Ethernet adapter, the web sites they access, and some of the messages they exchange.

Bob Bosen

Bob Bosen began building personal computers in 1969, and he had already completed and programmed three of his own machines before Jobs and Wozniak revealed the "Apple 1". He invented modern one-time password systems in 1979 and holds corresponding patents in the US and UK. His "SafeWord System" is in widespread use throughout the world, providing strong authentication for millions of network users every day. He frequently uses Wireshark to troubleshoot and research network applications, and he publishes the well-known "AskMisterWizard.com" online video magazine.

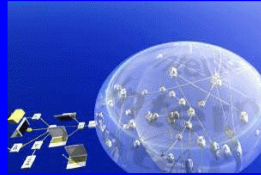
If you like Hakin9, you're ready for the animated video clips you'll find at:

www.AskMisterWizard.com

your online technology video magazine



How Ethernet works



How the Internet works



Hubs, Switches, and Routers,
and how they work



How to configure all kinds of network
equipment



How to use, configure, and abuse
Wireless Ethernet Bridges and
other advanced WiFi gear



Traceroute: "Ping on Steroids"

Sometimes you need more than a static image with a caption and surrounding text. When you want beautifully animated movie clips showing how technology really works, you want AskMisterWizard.com.

Each of our monthly publications features detailed video clips with clear, concise explanations of modern technology. Please join us today!

Using Wireshark to Analyze a Wireless Protocol

Wireshark is the perfect platform to troubleshoot wireless networks. In this tutorial, I will demonstrate how to support a new wireless

protocol in Wireshark. A wireless protocol in the real world is very complicated, so I will use ASN.1 technology to generate the source code of a dissector. Some advanced topics, such as export information, tap listeners, and so on, will be briefly introduced. Protocol analysis is extremely important, both for engineers in developing a complicated communication system, or for network supervision and fault diagnosis. Wireless networking is a bit more complex than a wired one. Countless standards, protocols, and implementations causes trouble for administrators trying to solve network problems. Fortunately, Wireshark has sophisticated wireless protocol analysis support to troubleshoot wireless networks.

In this article, we'll try to demonstrate how to analyze the real-world captures of a wireless communication protocol, *TERrestrial TRunked RADio* (TETRA). We will discuss how to sniff the wireless data and to dissect the protocol data.

TETRA Protocol Stack

TETRA is a specialist Professional Mobile Radio specification approved by ETSI. TETRA was specifically designed for use by government agencies, emergency services, rail transportation staff, transport services and the military. TETRA requires fast call set-up times (<0.5s), and since most call durations last less than 1 minute, the operations of channel assignment and release are frequent.

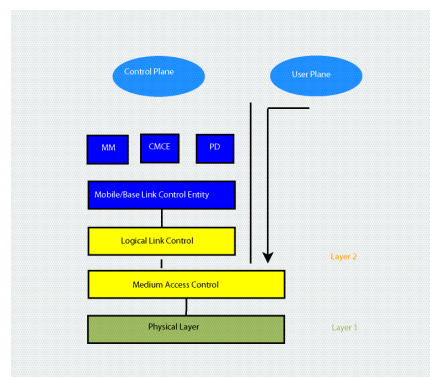


Figure 1. TETRA V+D Air Interface protocol stack

The TETRA Voice plus Data Air Interface (V+D AI) protocol stack is shown in Figure 1. The base of the protocol stack rests on the physical layer. The data link layer is composed of two sub-layer entities (MAC and LLC). An explicit Medium Access Control (MAC) sub-layer is introduced

to handle the problem of sharing the medium by a number of users. At the MAC, the protocol stack is divided into two parts, the user plane (U-plane), for transporting information without addressing capability, and the control plane (C-plane), for signaling and user data with addressing capability. A *Logical Link Control* (LLC) resides above the MAC and is responsible for controlling the logical link between a MS and a BS over a single radio hop. An explicit Mobile/Base Control Entity (MLE/BLE) sub-layer resides above the LLC for handling establishment and maintaining the connection to the BS. The MLE/BLE also acts as a convergence, so the same layer 3 entities could be used on top of different layer 2 entities. At the top of the protocol stack (layer 3), several entities may be present: *Mobility Management* (MM), *Circuit Mode Control Entity* (CMCE) and TETRA *packet data protocol* (PD). The interactions between layers go through *Service Access Points* (SAPs).

Capture wireless data

We need a hardware device to capture the traffic from the air and send it to Wireshark, that then decodes the traffic data into a format that helps administrators track down issues.

The primary motive for using Wireshark to analyze TETRA protocol data, is to help us develop our base station (BS) and mobile switch center (MSC) of TETRA. Figure 2 shows a diagram of our system architecture. A TETRA BS includes TETRA layer 1 and layer 2. The MAC itself is divided into two sub-layers, the upper and lower MAC. The lower MAC performs the channel coding, interleaving and scrambling. The upper MAC performs the other MAC protocol functions. In our system, an FPGA is used to implement the features of physical layer (PL) and the lower MAC (LMAC), while Base Station Controller (BSC) provides the functions of the upper MAC and LLC layers. TMV-SAP inside the MAC layer allows a protocol description using primitives and logical channels. By using the TMV-UNITDATA request primitive, the C-plane or U-plane information provided by higher layers will be placed into the appropriate logical channel and transmitted to the physical layer in the assigned timeslot, in the multiple frames. When lower MAC receives the data from an MS, it will send the data to upper MAC using TMV-UNITDATA indication primitive.

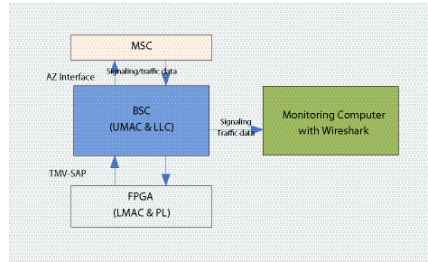


Figure 2. System architecture of TETRA BSC and MSC

There is no TETRA standard between a BS and an MSC, so we define this interface as AZ Interface in our system, just like A-Interface in GSM or Iu Interface in UMTS. A BSC connects to an MSC via Ethernets, and exchanges signaling using UDP protocol. U-Plane traffic data will be transferred using Real-time Transport Protocol (RTP) among TETRA networks. RTP provides mechanisms for the sending and receiving applications to support streaming data, so we choose RTP protocol to transfer traffic data in our system like most VoIP systems.

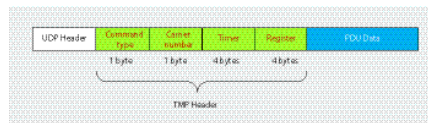


Figure 3. The packet format of TMP

BSC forwards all signaling and U-plane data, exchanged at both AZ Interface and TMV-SAP, to a monitoring computer for the purpose of observation and analysis. We defined the format of the TMV-SAP data as *TETRA Monitor Protocol* (TMP). This protocol will be discussed in a later section. Wireshark will be installed in the monitoring computer to capture and save the packet data. Because all the signaling and U-plane data is not standardized, we need to develop custom dissectors to analyze the captured data.

Another choice to capture the wireless TETRA data is using Osmocom TETRA. Osmocom TETRA project is an open source Software Defined Radio TETRA Air interface sniffer, which aims at implementing the sending and receiving part of the TETRA MAC/PHY layer.

Currently, Osmocom TETRA project can

- receive, demodulate and decode TETRA downlink signals of real-world TETRA networks
- display information about SYNC, SYSINFO, MM and CMCE PDUs

- forward those TETRA downlink signals to the Wireshark protocol analyzer
 - forward IP packets contained in TETRA SNDCP to a local tun/tap device
- Osmocom TETRA also adopts our TETRA Monitor Protocol.

TETRA Monitor Protocol

TETRA Monitor Protocol (TMP) is used to collect the information from TMV-SAP of a TETRA base station. TMP is based on UDP protocol and the target port number is 7074. Each TMP packet contains only one TETRA burst. The packet format for TMP data is defined in Figure 3. The *Command type* field indicates the nature of the follow-up data in the monitoring message, which is defined in Table 1. *MAC-Timer* is not a primitive defined in the TETRA standard, and it is used to help software developers to process the interrupt of the time slot. *TMV-UNITDATA indication Done* and *TMV-UNITDATA request Done* are similar to *TMV-UNITDATA indication* and *TMV-UNITDATA request* primitives, which are conducive to software debugging.

Table 1. Command type field information element contents

Command type	Meaning	Remark
1	TMV-UNITDATA request	The BS sends the data to an MS.
2	TMV-UNITDATA indication	An MS sends the data to the BS.
3	MAC-Timer	No data to be sent or received
127	TMV-UNITDATA indication Done	This message will be sent by a base station after the data are written to the LLC layer.
128	TMV-UNITDATA request Done	This message will be sent by a base station after the data are written to the lower MAC layer.

Carrier number field is used to distinguish different carrier.

Table 2. Bit description of Timer field

BIT	Symbol	Description
5:0	MFN	multiple frame number
10:6	FN	frame number
12:11	SN	Slot number
31:13	Reserved	

TETRA is a TDMA system, and hence Timer field contains the time slot information about the packet. The bit description of Timer field is shown in Table 2.

The meaning of Register field depends on the value of the *Command type* field. The bit description of the *Register* field of *TMV-UNITDATA request* and *TMV-UNITDATA indication* primitive are respectively shown in Table 3 and Table 4.

Table 4. The bit description of Register field of TMV-UNITDATA indication primitives

BIT	Symbol	Value	Description
1:0	LCHN	01	1 logical channel
		10	2 logical channels
		Reserved	Reserved
2	CRC1	0	OK
		1	Error
3	CRC2	0	OK
		1	Error
7:4	FLCHTP (First logical channel)		See Table 5
11:8	SLCHTP (Second logical channel)		See Table 5
31:12	Reserved	Reserved	Reserved

Table 5. Logical channel type information element contents

Logical Channel type	Meaning
1	AACH
2	SCH/F
3	SCH/HD
5	BSCH
6	BNCH
7	TCH/F
8	TCH/H
9	TCH/2.4
10	TCH/4.8
11	STCH
12	TCH/7.2
15	SCH/HU
Others	Reserved

Writing Wireshark Dissectors

Dissectors are what allow Wireshark to decode individual protocols and present them in readable format. We developed three Wireshark dissectors, TMV-SAP dissector, AZ Interface dissector and TETRA traffic dissector, for deep analysis of the TETRA protocol.

- TMV-SAP dissector will decode all the parameters of TMV-SAP primitives, including time slots, logical channel type and data, and so on.
- AZ Interface dissector will decode all the parameters of TLA-SAP, TLB-SAP and TLC-SAP primitives.
- Wireshark provides a built-in dissector for RTP, but RTP payload types defined in RFC 3551 do not include TETRA traffic data, so the default RTP dissector can't identify our TETRA traffic data. We need to write a TETRA traffic dissector to solve this problem.

Both TMV-SAP dissector and AZ Interface dissector are registered as the dissector of "udp.port". TETRA traffic dissector is a sub-dissector of "rtp.pt", and it will decode all parts of TETRA traffic data except the RTP protocol header.

TETRA TMV-SAP dissector is integrated into the official release of Wireshark since version 1.6 and you can view the complete source code of TMV-SAP dissector in the source code package. The implantation details of the other two dissectors are outside the scope of this article.

A protocol dissector can be written in C or Lua. Lua is a powerful light-weight programming language designed for extending applications. Although it's possible to write dissectors in Lua, most Wireshark dissectors are written in C, because it is several times faster. You can use Lua for prototyping dissectors, as during reverse engineering, you can save time for finding out how things work.

Wireshark also supports the implementation of protocol dissectors as plug-ins. Plug-ins can be developed and debugged without having to rebuild the whole Wireshark distribution. Under Windows, you can compile a plug-in into a *.DLL* file and place it into *C:\Program Files\Wireshark\plugins/<VERSION NUMBER>* directory. Wireshark will automatically load all plug-ins when it starts.

The first step in the development process is to acquire the Wireshark source code. The source code of Wireshark including all protocol dissectors can be done directly from the Wireshark website by hovering over the Develop link and clicking 'Browse the Code'. This link will send you to the Wireshark subversion repository, where you can view the current release code for Wireshark as well as the code for previous releases. Several open source libraries and tools are required for compiling the source code of the Wireshark dissector, so it is inconvenient to configure the build environment. If you are developing a Wireshark dissector under Windows, please refer to Ken Thompson's excellent article, "*Creating Your Own Custom Wireshark Dissector*", which is published on the *Code Project* web site. You can find detailed step by steps required to configure the build environment. You can also find a lot of useful information about the Wireshark build environment on other OS' at www.wireshark.org website. We need to create a *proto_register_tetra* function that was registered with Wireshark for our packet dissection.

Listing 1. *The code of proto_reg_handoff_tetra function*

```
537 void proto_reg_handoff_tetra(void)
538 {
539     static gboolean initialized=FALSE;
540
541     if (!initialized) {
```

```
542 data_handle = find_dissector("data");
543 tetra_handle = create_dissector_handle(dissect_tetra, proto_tetra);
544 dissector_add_uint("udp.port", global_tetra_port, tetra_handle);
545 }
546
547 }
```

The *proto_reg_handoff_tetra* function is used to instruct Wireshark on when to call your dissector (Listing 1). The *create_dissector_handle* function passes the function that Wireshark calls to dissect the packets and the *proto_xxx* value that was registered as the protocol in the *proto_register_protocol* function. The *dissector_add* function will trigger Wireshark to pass only the packet of UDP port 7074 to our dissector. When Wireshark receives a packet met with the criteria specified in the *proto_reg_handoff_tetra* function, it will call *dissect_tetra* and pass three important data structures to this function: *tvb*, *pinfo*, and *tree*.

- The *tvb* structure is used to extract and decode the data contained in each element of the packet.
- The *pinfo* structure provides specific information about the packet, based on information that was previously dissected by other processes (e.g., the *pinfo* structure tells you which packet number each relates to). It also contains flags for processing fragmented packets or multiple dissections.
- The *tree* structure provides a pointer towards the location in memory of the protocol tree data.

Please refer to the *README.developer* document located in the *doc* directory of the Wireshark source code package for further information related to dissector development.

Generate the dissector from ASN.1

As previously mentioned, a protocol dissector is commonly written in C, but Wireshark also provides the *Asn2wrs* compiler which generates the C source code of a dissector from an Abstract Syntax Notation One (ASN.1) specification of a protocol. ASN.1 is an international standard and provides flexible notation that describes rules and structures for representing, encoding, transmitting, and decoding data in telecommunications and computer networking. The *Asn2wrs* compiler is still a work in progress but has been used to create a number of dissectors. Next, we will use ASN.1 to develop the TMV-SAP dissector.

Table 63: D-CONNECT PDU contents

Information element	Length	Type	Owner	C/O/M	Remark
PDU Type	5	1	CC	M	
Call identifier	14	1	CC	M	
Call time-out	4	1	CC	M	
Hook method selection	1	1	CC	M	
Simplex/duplex selection	1	1	CC	M	
Transmission grant	2	1	CC	M	
Transmission request permission	1	1	CC	M	
Call ownership	1	1	CC	M	
Call priority	4	2	CC	O	
Basic service information	8	2	CC	O	See note
Temporary address	24	2	CC	O	
Notification indicator	6	2	SS	O	
Facility		3	SS	O	
Proprietary		3	-	O	
NOTE: If different from requested.					

Figure 4. An example of PDU description in TETRA standards

The TMV-SAP dissector will decode all three layers of PDUs, both uplink and downlink, and which remarkably improves the efficiency of debugging the AI protocol. The biggest challenge is the complex PDU encoding rule of TETRA. The TETRA protocol is defined using a tabular notation, to identify fields in the encoding structure (Figure 4), supplemented by English language text to define the encoding of those fields. The listed fields include both those carrying application semantics (that are relevant to an application programmer) and also determinant fields (that are relevant only to encoding/decoding code). Thomas Weigert and Paul Dietz pointed out that TETRA PDUs can't be expressed in ASN.1 syntax, so they designed a specific language and code generator for PDU decoding, only available in Motorola for internal use. With carefully investigation, we find that although the rule of TETRA does not accord with any existing ASN.1 encoding rules. However, it is very close to the UNALIGNED PER rule of ASN.1 (except from some uncommon features, such as Type 3 elements), so most TETRA PDU still can be processed by Asn2wrs compiler in Wireshark.

PDU decoding using ASN.1

Three different types of fields may be contained in a TETRA PDU.

Type 1 fields are mandatory and are therefore always present. They can be simply defined one by one in ASN.1 file with proper data type.

After all type 1 fields, a TETRA PDU will contain a bit, referred to as the O-bit, indicating whether any more bits will follow. O-bit-optional can also be expressed by a CHOICE type, where the first element is NULL type, and the second element is a SEQUENCE type, of all Type 2 fields. An example of O-bit-optional is shown as follows.

```

.....
optional-elements CHOICE
{
no-type2 NULL,
type2-parameters SEQUENCE {
....
}
}
}
.....

```

Type 2 fields, in a TETRA PDU, are optional. The presence of each such field is indicated by a flag bit, referred to as the P-bit. While the Type 2 field itself may be missing, its correlated P-bit will always be present (provided that the O-bit indicates that there are any following bits). Type 2 fields may be omitted but their order cannot be changed. Similar to O-bit-optional, Type 2 fields can also be expressed by a CHOICE type. Following is an example of Type 2 field.

```

.....
called-party-mnc CHOICE {
none NULL,
called-party-mnc INTEGER ( 0..16383)
},
.....

```

Listing 2 is a complete example of a TETRA PDU with Type 1 and Type 2 fields expressed in ASN.1 notation. Figure 5 is the decoding result displayed in Wireshark.

Listing 2. D-CONNECT PDU expressed in ASN.1 notation

```

2130 D-CONNECT ::=
2131 SEQUENCE{
2132 call-identifier INTEGER (0..1023),
2133 call-time-out INTEGER (0..31),
2134 hook-method-selection BOOLEAN,
2135 simplex-duplex-selection ENUMERATED {simplex(0), duplex(1)},
2136 transmission-grant INTEGER (0..3),
2137 transmission-request-permission INTEGER (0..1) ,
2138 call-ownership INTEGER (0..1) ,
2139 optional-elements CHOICE{
2140 no-type2 NULL,
2141 type2-parameters SEQUENCE {
2142 call-priority CHOICE{none NULL, call-priority INTEGER (0..15)},
2143 basic-service-information CHOICE{none NULL, basic-service-information Basic-
service-information},
2144 temporary-address CHOICE { none NULL, temporary-address Calling-party-address-
type},
2145 notification-indicator CHOICE { none NULL, notification-indicator INTEGER
(0..63)},
2146 prop [15] CHOICE {none NULL, prop [15] Proprietary }
2147 }
2148 }
2149 }

```

Asn2wrs Compiler

Asn2wrs Compiler is included in the source code package of Wireshark, which is written in Python. The compiler needs 4 input files; an ASN.1

description of a protocol, a .cnf file, and two template files. One template file is .c file, which includes the register and handoff function of the dissector. The other one is the header file (.h).

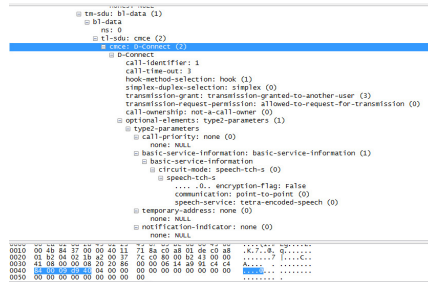


Figure 5. The decoding result of D-CONNECT PDU

In our TETRA dissector, we decode the TMV header part in the template file with manual codes and handle the PDU data using ASN.1 generated codes.

The .cnf file tells the compiler what to do with certain things, and to skip auto generation for some ASN1 entries. In Listing 3, we append a string about the PDU name to the INFO column of Wireshark Graphical User Interface (GUI) window when the code dissects a PDU. Put %(DEFAULT_BODY)s inside and #.FN_BODY will insert the original code there.

Listing 3. A block of code in .cnf file

```

113 #.FN_BODY D-CONNECT
114 %(DEFAULT_BODY)s
115 col_append_sep_str(actx->pinfo->cinfo, COL_INFO, NULL, "D-CONNECT");
116 #.END

```

Display Filters

In a busy TETRA system, the deluge of packets would be too much to handle. In this situation, Wireshark provides powerful display filters, so that users can specify which packets will be shown in Wireshark’s GUI. Because all of the packets are still in memory, they become visible when you reset your display filter.

Wireshark provides a simple but powerful display filter language that allows you to build quite complex filter expressions. You can use any filterable fields provided by our dissectors to sift through the display records. For example, if you want to find a setup of a voice call, you can simply enter tetra.u_Setup in the filter window. Table 6 shows some common display filters.

Table 6. Some display filters

Display filter	Filter expression
TMV-SAP primitives	tetra.timer

TMV-UNITDATA request primitive	tetra.txreg
TMV-UNITDATA indication primitive	tetra.rvster
Both MAC-RESOURCE and MAC-ACCESS PDU	tetra.MAC_RESOURCE tetra.MAC_ACCESS
CMCE U-SETUP PDU	tetra.u_Setup
Uplink voice data (TCH/F)	tetra.rxchannel1 == 3
Downlink voice data	tetra.txchannel2 == 3

Further improvements

The TETRA dissector included in the official release of Wireshark provides the basic ability to analyze the TETRA AI protocol. We can use some advance features of Wireshark to improve the function of the TETRA dissector. In this section, we will show improvement in our dissector.

Expert information

Expert information is the log of “possibly interesting” behavior in a capture, which allows users to get a summary of what they might want to look at. Expert information will be recorded by calling *expert_add_info_format* API with an item to which expert info is attached during the packet dissection. Four severity levels are supported: Chat, Note, Warn and Error. For example, we can check the CRC (Cyclic Redundancy Check) value of all logical channels as follows:

```
if(!(rxreg >> (i + 2) & 0x01)) /* CRC is true */
{
.....
}
else
expert_add_info_format(pinfo, crc_item, PI_CHECKSUM, PI_WARN,
"The CRC of this channel is incorrect.")
```

If the CRC value is incorrect, the dissector will report it as a warning. From the expert information dialog in Figure – 6, we found 10 CRC errors, which is much higher than we would expect. All the errors were occurring on STCH (STealing CHannel). The STCH is a channel associated with a TCH (Traffic Channel) that temporarily “steals” a part of the associated TCH capacity to transmit control messages. With careful checking of these error packets, we found a tiny bug in the channel decoder.

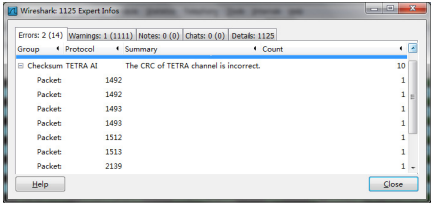


Figure 6. Error message shown in Expert Information dialog

Tap listener

The tap system is a powerful and flexible mechanism to get event driven notifications on packets matching certain protocols and/or filters. In *proto_register_tetra* function, we can attach to taps provided by dissectors. Here is the example code:

```
stats_tree_register("tetra", /* the proto we are going to "tap" */
"tetra_terms", /* the abbreviation for this tree */
str, /* the name of the menu and window */
0,
tetra_stats_tree_packet, /* the per packet callback */
tetra_stats_tree_init, /* the init callback */
NULL ); /* the cleanup callback (in this case there isn't) */
```

In this example, *tetra_stats_tree_packet* function is the callback function of the tap listener, which will receive the data sent by taps.

Taps can supply pre-digested data to listeners via *tap_queue_packet* function, and then the tap listeners process data supplied by the taps.

Now, we will show an example about the channel load of *Main Control CHannel* (MCCH). In each TETRA cell, one RF carrier shall be defined as the main carrier. Whenever a MCCH is used, it is located on the timeslot 1 of the main carrier. MCCH is very important for the TETRA system. The MCCH is used for signaling related to the setup of voice calls that are then performed on TCH. In the TETRA system, the *Short Data Service* (SDS), similar to short message service in GSM, also uses the MCCH. Hence, in cases of extremely high SDS traffic activity in a cell, the voice call could be blocked due to the collision in random access. We have to monitor the uplink channel load of MCCH.

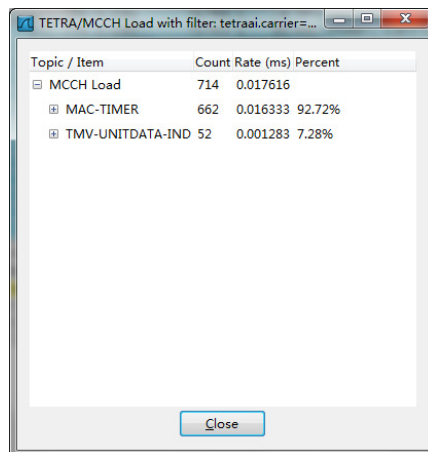


Figure 7. Statistics of channel load of MCCH

Figure 7 is a running test of the uplink channel load of MCCH. MAC-TIMER indicates no uplink load, while TMV-UNITDAT-IND means that some MSs send the signaling or data to MCCH. In this test, the uplink only

loads about 7.28%, and this is relatively low. If the channel load of MCCH is higher than 50%, we need to take some actions such as, for instance, adding a SCCH to the cell.

On the Web

- <http://www.codeproject.com/Articles/19426/Creating-Your-Own-Custom-Wireshark-Dissector> – A guide for developer WireShark dissector under Windows
- <http://tetra.osmocom.org/trac/> – The Osmocom TETRA project
- <http://www.itu.int/ITU-T/asn1/introduction/index.htm> – Introduction to ASN.1

LI Hai

LI Hai is an associate professor of Beijing Institute of Technology (BIT). He is the leader of Professional Mobile Communication Research Group of BIT. He has led his team to develop a base station and switch system of the TETRA system, including both hardware devices and software protocol stacks. His team also provides the world's first automatic TETRA interoperability test system based on TTCN-3. His research interests include embedded operating systems, real-time systems, and protocol engineering of wireless communication systems. You can reach him at haili@bit.edu.cn.

Capturing WiFi traffic with Wireshark

For many years, Wireshark has been used to capture and decode data packets on wired networks. Wireshark can also capture IEEE 802.11 wireless traffic while running on a variety of operating systems.

This article describes how Wireshark is used to capture / decode 802.11 traffic and its configuration specifics based on the operating system you are running. It covers three popular OS: MS-Windows, Linux and OS X. It also covers two ways to indirectly collect 802.11 traffic and then analyze it with Wireshark.

Wireshark on Windows

Wireshark in conjunction with AirPcap will enable you to capture 802.11 traffic on Microsoft Windows platforms. AirPcap is a WiFi USB adapter from Riverbed (formerly CACE Technologies). It provides a wireless packet capture solution for MS Windows environments. AirPcap captures full 802.11 data, management and control frames that can be viewed in Wireshark, providing in-depth protocol dissection and analysis capabilities. AirPcap is available in three models: AirPcap Classic, AirPcap Tx and AirPcap Nx. All models can perform packet capture and both the Tx and Nx models can also do packet injection. Pricing varies from \$198 to \$698.

Please note that AirPcap Classic and Tx only support 802.11b/g whereas AirPcap Nx supports 802.11a/b/g/n (Figure 1).



Figure 1. Wireshark Multi Pack

AirPcap setup is easy. Its USB adapter requires a special driver to be installed in Windows. This can be done from the provided CD by selecting ‘install driver’ at the install dialog. Depending on the Windows operating system version, when you plug the adapter in for the first time, Windows may show the “Found New Hardware Wizard”. From that same CD, you can also install Wireshark for Windows.

Once the driver installed, the new adapter will display in AirPcap control panel as “AirPcap USB wireless capture adapter nr 00”. Zero meaning the first adapter, 01 the second adapter and so on.

An AirPcap adapter will capture on one channel at a time. AirPcap control panel also enables you to select the channel on which the adapter will capture packets. If you purchased the multi-channel version, the control panel will display “AirPcap Multi-channel Aggregator”. Using 3 USB adapters, AirPcap enables Wireshark capturing simultaneously on 3 channels. For instance, channels 1, 6 and 11 in the 2.4 GHz band.

A special *wireless toolbar* appears in Wireshark when at least one AirPcap adapter is plugged into one of the USB ports, and can be used to change the parameters of the currently active wireless interfaces. This is where you can select to frame decryption for WEP or WPA/WPA2.

The AirPcap driver can use a set of WEP keys to decrypt traffic that encrypted with WEP. The list of keys can be edited by selecting the Keys tab in the AirPcap control panel. The AirPcap driver will attempt to decrypt the WEP encrypted frame using the your supplied set of WEP keys. That is, the driver will try all of the WEP keys for each frame until it finds one that decrypts the frame. By configuring the AirPcap driver with several WEP

keys, it is possible to decrypt traffic coming from multiple WiFi access points that are using different WEP keys.

Decryption of WPA/WPA2 can be done by Wireshark by setting the wireless toolbar decryption mode to Wireshark. In this mode, the driver doesn't perform any decryption of the captured packets (as in the case of WEP), and they are decrypted by Wireshark while displaying them. In order to decrypt WPA and WPA2 you will need to configure the pre-shared key and capture the 4-way EAPOL handshake used to establish the pairwise transient key (PTK) used for a session. Wireshark can only decrypt "WPA personal" sessions, which use pre-shared keys. Decryption of "WPA Enterprise" sessions is not supported.

Finally, one nice feature about AirPcap Nx adapter hardware: it has two internal antennas and two integrated MC-Card connectors for optional external antennas allowing you to do long-range capture. External antennas can be either omnidirectional or directional.

References

- AirPcap Home Page – http://www.riverbed.com/us/products/cascade/wireshark_enhancements/airpcap.php
- AirPcap Products Catalog – Pricing – <http://www.cacotech.com/products/catalog/>

Wireshark on MAC OS X

Capturing 802.11 frames with Wireshark under OS X can be achieved using your MacBook built-in WiFi adapter. The following discussion relates how it was setup with OS X Lion. This may vary with other versions. Open a terminal window and set permissions on the BPF devices (Berkeley Packet Filter) so they can be accessed in read and write mode:

```
# sudo chmod 666 /dev/bpf*
```

The above `sudo` command requires you provide your account password. Verify whether the BPF devices are correctly set: Listing 1.

Listing 1. Setting BPF devices

```
# ls -l /dev/bpf*  
crw-rw-rw- 1 root admin 23, 0 4 Oct 06:31 /dev/bpf0  
crw-rw-rw- 1 root admin 23, 1 4 Oct 06:31 /dev/bpf1  
crw-rw-rw- 1 root admin 23, 2 4 Oct 06:31 /dev/bpf2  
crw-rw-rw- 1 root admin 23, 3 4 Oct 06:31 /dev/bpf3
```

Next, create a symbolic link to the *airport* utility, this will prevent you from typing the whole path every time:

```
# ln -s sudo /System/Library/PrivateFrameworks/
```



```
Apple80211.framework/Versions/Current/Resources  
/usr/sbin/airport
```

Now, with the *airport* utility, disassociate your WiFi adapter and set it to the channel you want to capture. In the following example the `-z` flag will disassociate your NIC and flag `-c 11` sets the channel to 11.

```
# sudo airport -z -c 11
```

To verify whether your channel is set correctly, type `airport -I` and check the last line of the output: Listing 2.

Listing 2. *Verifying your channel*

```
# airport -I  
agrCtlRSSI: -73  
agrExtRSSI: 0  
agrCtlNoise: -91  
agrExtNoise: 0  
state: running  
op mode: station  
lastTxRate: 18  
maxRate: 54  
lastAssocStatus: 0  
802.11 auth: open  
link auth: wpa2-psk  
BSSID: 10:84:d:e4:b8:7f  
SSID: xtnet  
MCS: -1  
channel: 11
```

Next, download and install Wireshark for OS X at:

<http://www.wireshark.org/download.html>.

Start Wireshark. From the *Capture Options* make sure your WiFi adapter will be listed as *en1 802.11 plus Radiotap Header* and it must be *enabled*. Also, ensure you check *Capture all in promiscuous mode*.

You are all set to go and can start capturing WiFi on interface *en1*.

Optionally, you can add a new column display channel & frequency. To do so, right click any column heading in Wireshark OS X, select *Column Preferences*, click the *Add* button and select *Frequency/Channel* from the *Field Type* pull-down list. Also rename that new column to something meaningful (e.g., channel).

Note

The *airport* utility can also be used to display nearby access points: Listing 3.

Listing 3. *The airport utility displaying access points*

```
# airport -s  
SSID BSSID RSSI CHANNEL HT CC SECURITY (auth/unicast/group)  
linksys 00:18:f8:ef:93:af -87 6 N -- NONE  
bing 10:c8:d0:1a:e4:f3 -90 10 Y CA WPA2(PSK/AES/AES)  
NETGEAR 00:0f:b5:5d:06:0c -89 11 N -- WPA(PSK/TKIP/TKIP)  
BELL789 c0:83:0a:53:b7:41 -88 11 N US WEP
```

```
lolo 00:22:b0:d2:63:67 -89 1,+1 Y -- WEP
xxtnet5 10:84:0d:f4:c8:80 -63 36,+1 Y CA WPA(PSK/TKIP/TKIP) WPA2(PSK/AES,TKIP/TKIP)
xxtnet 20:54:4d:d4:98:4f -64 11 N CA WPA(PSK/TKIP/TKIP) WPA2(PSK/AES,TKIP/TKIP)
Belkin 00:1c:df:39:81:f6 -84 11 N -- WPA(PSK/TKIP/TKIP)
```

You can repeat the above command in a loop as you walk/survey with your MacBook:

```
# while true; do airport -s; sleep 1; done
```

To stop it, type *control-c*.

Wireshark on Linux

Wireshark can run on several Linux distributions. In order to capture / decode 802.11 frames, you need to set your WiFi adapter into promiscuous mode and use Wireshark from that point. That procedure varies from one WiFi adapter vendor to another.

One way to help achieving this is through the *airmon-ng* utility from the *aircrack-ng* suite. It can be installed on the Linux variant you prefer. You will find convenient to use the *BackTrack* Linux distribution. BackTrack is already loaded with hundreds of tools for penetration testing, security analysis, etc. And it already has both *aircrack-ng* and *Wireshark* installed. You can download the BackTrack .iso file, burn it onto a DVD and boot from that DVD.

BackTrack can later be installed on your hard drive. Even better, install BackTrack on a persistent USB thumb drive and use it to run BackTrack from any laptop that can boot from a USB. With this portable Linux solution, your scripts, test cases, configurations, etc. will be preserved from one boot to another. For more details on how to create a persistent USB for BackTrack, please visit the link listed in the references below.

Airmon-ng creates a new network interface which is automatically configured to operate in promiscuous mode (or monitor mode). Please note that the *Aircrack-ng* suite will work with several WiFi adapters that are shipped with the laptops and external USB WiFi adapters. A compatibility list is available here: http://www.aircrack-ng.org/doku.php?id=compatibility_drivers.

Once you have a WiFi adapter capable of capturing, you can use Wireshark to capture and decode the 802.11 traffic. You can check the interfaces status by typing `airmon-ng`:

```
# airmon-ng
Interface Chipset Driver
eth1 Intel 2200BG ipw2200
```

The eth1 interface above is the built-in Intel WiFi adapter. We now insert the ALFA USB wireless adapter and invoke `airmon-ng` again. In the following example, we use an external WiFi USB adapter. Its model is ALFA AWUS036EH, 802.11b/g and WPA/WPA2 compliant. It uses a 5 dBi external antenna. Its chipset is a Realtek 8187 and it is packet injection capable.

```
# airmon-ng
Interface Chipset Driver
eth1 Intel 2200BG ipw2200
wlan0 RTL8187 rtl8187 - [phy0]
```

Notice that Linux OS named this interface `wlan0` and the ALFA USB adapter rtl8187 chipset is revealed. Now we set interface `wlan0` into promiscuous mode and we specify channel 11:

```
# airmon-ng start wlan0 11
Interface Chipset Driver
eth1 Intel 2200BG ipw2200
wlan0 RTL8187 rtl8187 - [phy0]
(monitor mode enabled on mon0)
```

the above command confirms that `wlan0` is now in monitor mode (promiscuous). If you type `airmon-ng` again, you will notice a new `mon0` interface:

```
# airmon-ng
Interface Chipset Driver
eth1 Intel 2200BG ipw2200
wlan0 RTL8187 rtl8187 - [phy0]
mon0 RTL8187 rtl8187 - [phy0]
```

Now start Wireshark and from *Capture > Interfaces > mon0 > Options* ensure that you checked Capture packets in *promiscuous* mode (this is the default value).

You can now start capturing on interface `mon0`. Wireshark will capture 802.11 traffic on channel 11 since it was specified in the previous `airmon-ng` command.

Note

To add the *channel* column in Wireshark Linux, proceed as follows: *Edit > Preferences > User Interface > Columns*.

Click *New* and enter a meaningful name in the *Title* field. Then select *Frequency/Channel* from the Format pull-down list. Adjust the column order using the *Up* and *Down* buttons. If you need to change channels, use the `iwconfig` command:

```
# iwconfig mon0 channel 6
```

The above will cause Wireshark to start capturing on channel 6. There is no need to stop Wireshark while doing this.

It is possible that the channel you set using *iwconfig* doesn't take effect.

This might happen if your WiFi adapter is associated to an access point. To prevent this, stop your networking daemon:

```
# sudo /etc/init.d/networking stop
```

You may want to enable networking later when you are done with sniffing:

```
# sudo /etc/init.d/networking start
```

Rebooting Linux will remove the `mon0` interface you created earlier with `airmon-ng`. But you can also remove `mon0` as follows:

```
# airmon-ng stop mon0
```

References

- BackTrack Home Page – <http://www.backtrack-linux.org/>
- BackTrack Persistent USB – http://www.backtrack-linux.org/wiki/index.php/Persistent_USB
- Aircrack-ng Home Page – <http://www.aircrack-ng.org/>

Wireshark and Kismet

Kismet is an 802.11 layer2 wireless network detector, sniffer, and intrusion detection system. Kismet will work with any wireless card which supports raw monitoring (rfmon) mode, and (with appropriate hardware) can sniff 802.11b, 802.11a, 802.11g, and 802.11n traffic. Every time you launch Kismet, it will create a whole set of new files. For instance:

```
# ls kismet*
Kismet-20121004-13-37-22-1.alert
Kismet-20121004-13-37-22-1.gpsxml
Kismet-20121004-13-37-22-1.nettxt
Kismet-20121004-13-37-22-1.netxml
Kismet-20121004-13-37-22-1.pcapdump
```

Kismet captures 802.11 frames in the file with extension `.pcapdump`. To ensure files are unique, Kismet prefixes them as follows: `Kismet-yyymmdd-hh-mm-ss-sequence#`.

While using Kismet to perform WiFi network analysis, 802.11 frames are collected on various channels. By default, Kismet is configured to do *channel hopping*. That is, Kismet will capture some 802.11 frames on channel 1, then will move to channel 6 and collect some frames, and then move to channel 11, etc. If you need to focus on a specific channel (e.g., channel 11), you can easily change this from the Kismet GUI as follows:

```
Kismet > Config Channel
default is (*) Hop
set it to (*) Lock and set Chan/Freq to 11
```

If you have the *aircrack-ng* suite installed, you can issue the `airmon-ng` command to examine the interfaces:

```
# airmon-ng
Interface Chipset Driver
eth1 Intel 2200BG ipw2200
wlan0 RTL8187 rtl8187 - [phy0]
wlan0mon RTL8187 rtl8187 - [phy0]
```

Above, are listed two physical interfaces, `eth1` with an Intel chipset and `wlan0` with a Realtek 8187 chipset. Kismet is currently configured to use `wlan0` for network analysis. After starting Kismet for a first time, it will create a *monitor mode* logical interface called `wlan0mon`. Kismet uses that interface to perform both network analysis and 802.11 frame capture.

The `iwconfig` command will also list the system interfaces. The following example shows two physical interfaces, `eth1` and `wlan0` along with logical interface `wlan0mon` (Mode:Monitor). As we previously locked the channel to 11, interface `wlan0mon` displays frequency 2.462 GHz which translates to channel 11. If you do not explicitly configure Kismet to lock in a specific channel, this will be reflected every time you execute the `iwconfig` command (the *frequency* value will vary constantly) (Listing 4).

Listing 4. *The usage of Kismet*

```
# iwconfig
lo no wireless extensions.
eth0 no wireless extensions.
eth1 unassociated ESSID:off/any
Mode:Managed Channel=0 Access Point: Not-Associated
Bit Rate:0 kb/s Tx-Power=20 dBm Sensitivity=8/0
Retry limit:7 RTS thr:off Fragment thr:off
Encryption key:off
Power Management:off
Link Quality:0 Signal level:0 Noise level:0
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0
wmaster0 no wireless extensions.
wlan0 IEEE 802.11bg ESSID:""
Mode:Managed Frequency:2.462 GHz Access Point: Not-Associated
Tx-Power=27 dBm
Retry min limit:7 RTS thr:off Fragment thr=2352 B
Encryption key:off
Power Management:off
Link Quality:0 Signal level:0 Noise level:0
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0
wlan0mon IEEE 802.11bg Mode:Monitor Frequency:2.462 GHz Tx-Power=27 dBm
Retry min limit:7 RTS thr:off Fragment thr=2352 B
Encryption key:off
Power Management:off
Link Quality:0 Signal level:0 Noise level:0
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
```

```
Tx excessive retries:0 Invalid misc:0 Missed beacon:0
```

After collecting 802.11 frames for a certain time, you can stop Kismet. Next, start Wireshark from the command line followed with the `.pcapdump` file name:

```
# wireshark Kismet-20121004-13-37-22-1.pcapdump
```

Or if you prefer, start Wireshark and then: File > Open > your `.pcapdump` file.

In case 802.11 frames are not decoded properly in Wireshark, check the `pcapdumpformat` parameter in Kismet configuration file `kismet.conf`. It is usually located under directory `/usr/etc`. You should see something similar to:

```
#pcapdumpformat=ppi  
pcapdumpformat=80211
```

By default, `pcapdumpformat` is set to `ppi`. Try commenting out `ppi` and uncomment `80211`. Restart Kismet, capture 802.11 frames for a while, then stop Kismet and use Wireshark to decode the newly created `.pcapdump` file.

References

- Kismet Home Page – <http://www.kismetwireless.net/>
- Kismet Documentation – <http://www.kismetwireless.net/documentation.shtml>

Wireshark and Cisco Lightweight AP

A Cisco LAP (*Lightweight Access Point*) is an enterprise AP that runs a lightweight IOS image (not to be confused with Apple iOS). Several enterprise LAPs will join a Cisco WLC (*Wireless LAN Controller*). LAPs then encapsulate all 802.11 client traffic in CAPWAP (RFC5415) frames and forward them to the WLC. This mode of operation is known as CUWN or Cisco Unified Wireless Networking.

Each LAP normally runs in *local* mode and forwards all client traffic to the WLC. You can configure a LAP in *sniffer* mode so it can capture 802.11 frames and forward them to a workstation that runs Wireshark. As a network administrator of several hundreds of LAPs, you can use Wireshark to sniff any LAP without having to travel to remote sites. In order to achieve this, you need to configure both the LAP and the Wireshark workstation.

LAP Configuration

From the WLC graphical interface, under the *Wireless* tab, select a LAP that you will dedicate as a sniffer. From the LAP *General* tab configure the *AP Mode* to *Sniffer*. The WLC will warn you that the LAP requires a reboot. Click on the OK button and wait a few minutes for the LAP to display again in the WLC user interface (Figure 2).

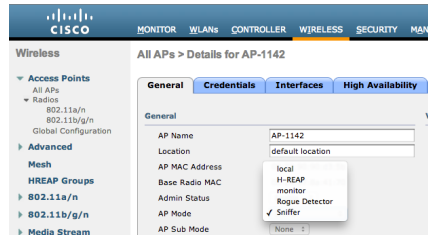


Figure 2. WLC Sniffer Mode

Next, from the *Wireless* tab, select the radio for which you need to capture traffic (802.11a/n or 802.11b/g/n) *Wireless* > *Access Points* > *Access Point Name* > *Radios 802.11a/n or 801.11b/g/n*.

Then, hover your mouse cursor on the blue triangle on the right and when the small pop-up displays, click *Configure* (Figure 3).

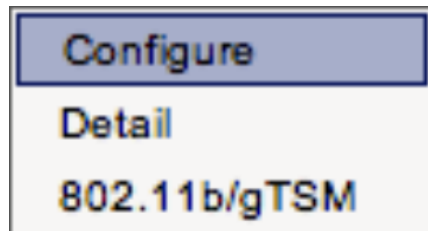


Figure 3. WLC Configure Radio

Under *Sniffer Channel Assignment*, check *Sniff*, then provide a channel on which to capture and then configure the *IP address* of the workstation running Wireshark. In the example below, the channel is set to *11* and the workstation is at IP *192.168.1.104* (Figure 4).



Figure 4. WLC Sniffer Channel

Wireshark Configuration

Start Wireshark on your *wired* workstation (e.g. at the IP address configured above).

Next, make sure you set Wireshark to decode for either *AIROPEEK* or *PEEKREMOTE*. This depends on the version of Wireshark you use. Starting with Wireshark 1.8.0, only *PEEKREMOTE* is available. These

decodes were originally developed for Airopeek / Omnippeek but also work with Wireshark. You will find more information about these decodes in the references section below (Figure 5).

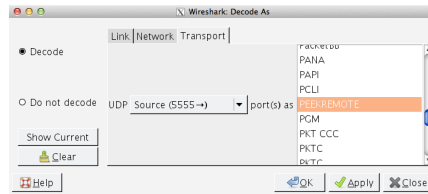


Figure 5. *Wireshark Peekremote Analyze > Decode As*

Transport Tab > UDP source (5555) AIROPEEK or PEEKREMOTE

Next, set the interface capture options to receive only traffic on UDP/5555. This filter is optional but strongly recommended as it excludes all the non-wireless related traffic from the capture. Consider that the WLC sends traffic to a UDP port there's no application listening on the sniffer side; this results in having a ICMP port-unreachable response for each packet received from the WLC.

Although this is expected, the filter above helps to exclude also this traffic which is useless and so it can only cause the trace to be bigger and more difficult to read.

Capture > Interfaces > Options

- double click the interface that will be used for capture
- set the *Capture Filter* box to: *udp port 5555* (Figure 6)

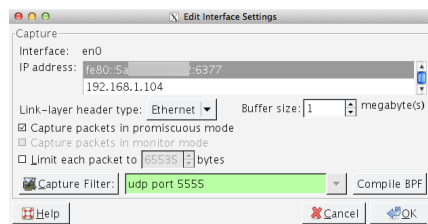


Figure 6. *Wireshark Capture Filter*

Wireshark now displays 802.11 traffic captured from the Cisco LAP. Whenever you are done with the capture, you can return to the WLC and reset the LAP configuration to *local* mode.

References

- CAPWAP RFC – <http://tools.ietf.org/html/rfc5415>
- Cisco Unified Wireless Networking – <http://www.cisco.com/en/US/products/hw/wireless/index.html>

- Wireshark Display Filter Reference –
<http://www.wireshark.org/docs/dfref/a/airopeek.html>;
<http://www.wireshark.org/docs/dfref/p/peekremote.html>

Conclusion

Wireshark remains a free / low-cost solution for capturing wireless frames. Wireshark can be used to capture and decode 802.11 WiFi traffic on a variety of operating systems. Third-party tools can collect WiFi traffic and save it in Wireshark readable format. Additionally, specialized hardware can capture 802.11 traffic and forward it directly to Wireshark for analysis. Depending on the operating system in use, you will need specific Wireshark / system configuration as well as appropriate hardware to get the job done.

STEVE WILLIAMS

Steve Williams is a freelance consultant with expertise in WiFi, Firewalls and Identity Management. Mr. Williams has been in the consulting business for the past 20 years. During that time, he tackled very large projects with major North American ISPs (Internet Service Providers), cable companies, manufacturing, banking. He also had the opportunity to consult and provide WiFi training to several enterprises, public and educational entities. Mr. Williams is the founder of Sudo Networks based in Montreal, Canada and he can be reached at info@sudonetworks.com.



HIGH-TECH BRIDGE[®]
INFORMATION SECURITY SOLUTIONS

www.htbridge.ch

ORIGINAL SWISS ETHICAL HACKING

Digital Forensics
Malware Analysis
Penetration Testing
Source Code Review
Security Audit & Consulting



GENEVA WORLD TRADE CENTER II T. +41 22 723 24 24 F. +41 22 788 35 71 info@htbridge.ch

Decoding and Decrypting Network Packets with Wireshark

In the article I will cover dissecting and decrypting Bluetooth High Speed over wireless traffic.

The main idea is that well known Bluetooth protocols, profiles and security mechanisms to be used with secondary radio are already present in many devices. Given that secondary radio is usually significantly faster we achieve faster data transfer while keeping existing API. The user does not need to worry about changing his code. See [1] for more details.

There are two flows of traffic during High Speed data transfers. One is coming through BR/EDR Bluetooth channel and the other through a wireless 802.11 interface. In this article decoding wireless traffic will be covered. Since an L2CAP connection is established through Bluetooth, the wireless dump lacks the connection signalling packets and therefore Wireshark cannot find out which protocol is in use on upper layers. Wireshark also needs Bluetooth the key to be able to decrypt wireless frames.

Encryption Basics

Connections between High Speed devices are encrypted and share symmetric keys. In 802.11 it has name Pairwise Transient Key. The PTK is generated by concatenating the following attributes: PMK, AP nonce (ANonce), STA nonce (SNonce), AP MAC address, and STA MAC address. Terminology 802.11 means: STA – station and AP – access point, for High Speed initiator and responder, a nonce is an arbitrary number used only once in a cryptographic communication. PMK is a shared secret key between two AMP controllers. It is valid throughout the whole session and needs to be exposed as little as possible. For more information see [3].

Getting Pairwise Master Key (PMK)

Bluetooth provides key material for wireless security by creating Dedicated AMP Link Key which is used by wireless devices as Pairwise Master Key. The PMK is needed for decrypting wireless encrypted frames.

After we pair two devices (SSP pairing is needed) bluetooth creates Bluetooth Link Keys (LK) which are usually stored. In Linux, the LK can be found in the following path:

```
/var/lib/bluetooth/<MAC Address>/linkkeys .
```

First we create Generic AMP Link Key (GAMP) given known LK.

$GAMP_LK = HMAC\text{-}SHA\text{-}256(LK||LK, 'gamp', 32)$ where $LK||LK$ means concatenations of 2 16 bits Link Keys forming 32 bit result array. Then we create Dedicated AMP Link Key.

$Dedicated_AMP_Link_Key = HMAC\text{-}SHA\text{-}256(GAMP_LK, '802b', 32)$. See [2] “Vol 2: 7.7.5 The Simple Pairing AMP Key Derivation Function h2” for more info.

The result PMK will be used by wireshark decryption engine after some modification below.

Decoding Bluetooth High Speed Traffic Over Wireless

Figure 1 shows captured wireless traffic taken with an external wireless card in monitor mode filtered by MAC addresses. We see two types of frames: LLC frames and 802.11 data which Wireshark was able to decode. Since we know that all High Speed frames shall have LLC headers we might assume that those frames without LLC headers are encrypted and that means that authentication and key generation is happening in packets marked as LLC.

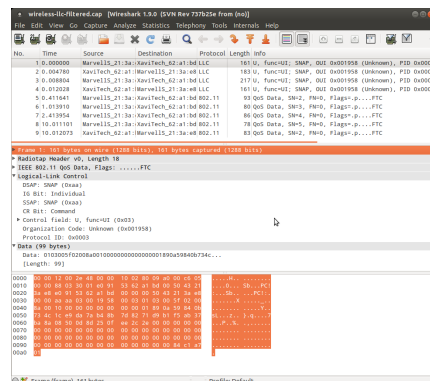


Figure 1. Captured wireless traffic

The Bluetooth specification specifies encapsulation methods used for data traffic in [2] “Vol 5: Table 5.1: 802.11 AMP LLC/SNAP encapsulation.” Wireshark already has LLC dissector and we only need to define our

Organization Unique Identifier (OUI) or Company Id and then register our OUI like it is shown in Listing 1.

Listing 1. Registration of Bluetooth OUI

```
#define OUI_BLUETOOTH 0x001958 /* Bluetooth SIG */
void proto_register_bt_oui(void)
{
    static hf_register_info hf[] = {
        { &hf_llc_bluetooth_pid,
          { "PID", "llc.bluetooth_pid", FT_UINT16, BASE_HEX,
            VALS(bluetooth_pid_vals), 0x0, "Protocol ID", HFILL }
        }
    };
    llc_add_oui(OUI_BLUETOOTH, "llc.bluetooth_pid", "Bluetooth OUI PID", hf);
}
```

Once complete, packets with Bluetooth OUI will be identified as Bluetooth High Speed packets. The field `llc.bluetooth_pid` identifies the type of data the packet contains. Listing 2 shows all possible data types.

Listing 2. Types of Bluetooth High Speed frames

```
#define AMP_U_L2CAP 0x0001
#define AMP_C_ACTIVITY_REPORT 0x0002
#define AMP_C_SECURITY_FRAME 0x0003
#define AMP_C_LINK_SUP_REQUEST 0x0004
#define AMP_C_LINK_SUP_REPLY 0x0005
static const value_string bluetooth_pid_vals[] = {
    { AMP_U_L2CAP, "AMP_U L2CAP ACL data" },
    { AMP_C_ACTIVITY_REPORT, "AMP-C Activity Report" },
    { AMP_C_SECURITY_FRAME, "AMP-C Security frames" },
    { AMP_C_LINK_SUP_REQUEST, "AMP-C Link supervision request" },
    { AMP_C_LINK_SUP_REPLY, "AMP-C Link supervision reply" },
    { 0, NULL }
};
```

What we have now is only LLC is dissected. The data coming after LLC header is dissected as raw data. We want Wireshark to dissect encapsulated frames from Wireshark's known protocols list since the tool already has almost all major protocol supported. For that we need to register dissectors of known protocols according to their `bluetooth_pid` values to LLC dissector table. AMP Security frames represents X11 Authentication which might be decoded by `eapol` dissector, AMP L2CAP ACL data frames might be decoded by `btl2cap` dissector.

Listing 3. Registering `eapol` and `btl2cap` dissectors

```
void proto_reg_handoff_bt_oui(void)
{
    dissector_handle_t eapol_handle;
    dissector_handle_t btl2cap_handle;
    eapol_handle = find_dissector("eapol");
    btl2cap_handle = find_dissector("btl2cap");
    dissector_add_uint("llc.bluetooth_pid", AMP_C_SECURITY_FRAME, eapol_handle);
    dissector_add_uint("llc.bluetooth_pid", AMP_U_L2CAP, btl2cap_handle);
}
```

Listing 3 shows adding L2CAP and EAPOL dissectors in the dissector table. First we find dissector handles with `find_dissector` and then we add handles with `dissector_add_uint`.

The change above allows Wireshark to decode EAPOL frames from the dump. Figure 2 shows Wireshark dissecting EAPOL frame, the first message in the 4-way authentication sequence.

After the EAPOL frames traffic is encrypted. This is because the authentication LLC header is also encrypted and those packets cannot be identified as Bluetooth High Speed data. We need to decrypt the packets and then Wireshark is able to understand the packet by looking at the decrypted LLC.

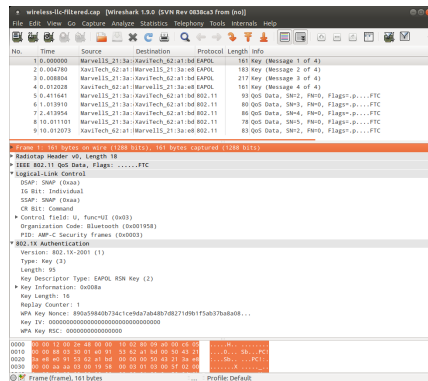


Figure 2. Decoding EAPOL packets

Decrypting Bluetooth Encrypted Data

Next step is to determine the decryption key. Fortunately we have all the required information like Bluetooth supplied PMK and trace containing the 4-way authentication. Wireshark already has the capability to derive *Pairwise Transient Key* (PTK) from a 4-way authentication sequence (shown as EAPOL in Wireshark) in the airpcap library.

Bluetooth EAPOL frames are not recognized because airpcap tries to only decode packets with special LLC header specifying type `0x88`, `0x8E` /* Type: 802.1X authentication */. The solution is to add second LLC header and filter only those two headers shown in Listing 4.

Listing 4. Adding second LLC header

```
file: epan/crypt/airpcap.c function: AirPCapPacketProcess
const guint8 bt_dot1x_header[] = {
0xAA, /* SSAP=SNAP */
0x03, /* Control field=Unnumbered frame */
0x00, 0x19, 0x58, /* Org. code=Bluetooth SIG */
0x00, 0x03 /* Type: Bluetooth Security */
};
```

```

/* Filter 802.1X authentication frames */
if (memcmp(data+offset, dot1x_header, 8) == 0 ||
memcmp(data+offset, bt_dot1x_header, 8) == 0) {

```

After this change airdpcap is able to find PTK key (given that PMK key is known by Wireshark through preferences) and then decrypt data traffic. Figure 3 shows.

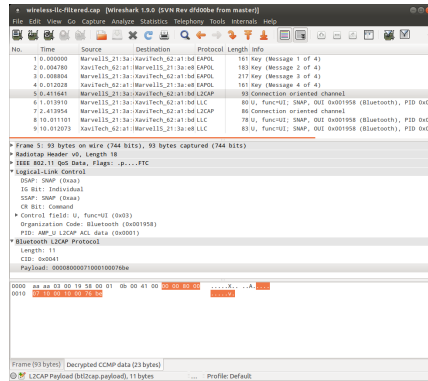


Figure 3. Decoding L2CAP packets in decrypted CCMP data

References

- [1] Bluetooth High Speed. <http://www.bluetooth.com/Pages/High-Speed.aspx>
- [2] BLUETOOTH SPECIFICATION Version 4.0 https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737
- [3] IEEE 802.11i-2004: Amendment 6: Medium Access Control (MAC) Security Enhancements <http://standards.ieee.org/getieee802/download/802.11i-2004.pdf>

Andrei Emeltchenko

Author has over 12 years of experience working with network protocols in Nokia, Nokia Siemens Networks and Intel.

Wireshark – Hacking WiFi Tool

Wireshark is cross-platform free and open-source packet analyzer. The project, formerly known as Ethereal started in 1998 and become the world’s foremost network protocol analyzer.

Gerald Combs, Ethereal’s creator, was unable to reach agreement with his now former employer, which holds trademark rights to the Ethereal name. Later, Wireshark was born. The current stable release of Wireshark is 1.8.3 at the time of writing this article. It supersedes all previous releases, including all releases of Ethereal.

When placed properly, Wireshark can be a great help for network administrator when it comes to network troubleshooting, such as latency issues, routing errors, buffer overflows, virus and malware infections analysis, slow network applications, broadcast and multicast storms, DNS resolution problems, interface mismatch, or security incidents.

As data streams flow across the network, the sniffer captures each packet and, if needed, decodes the packet's raw data. Depending on your needs, network data can be browsed via a GUI, or via the TTY-mode TShark utility. Importing traces from other programs such as tcpdump, Cisco IDS, Microsoft Network Monitor and others are also supported, so analyzing information from other sources is granted.

Capture Options

Wireshark is a really great tool when it comes to digging into large dump of wireless traffic. Capturing live network data is one of the major features. Before starting a packet capture, user should know answers to a simple question. Does my operating system supports mode I am going to use with my network interface? To answer this question please make some research about two of the six modes that wireless cards can operate in – Monitor mode and Promiscuous mode. In general Monitor mode only applies to wireless networks, while promiscuous mode can be used on both wired and wireless networks.

Monitor mode allows packets to be captured without having to associate with an access point or ad-hoc network. This mode may be used for malicious purposes such as passive packets sniffing, injecting packets to speed up cracking *Wired Equivalent Privacy* (WEP) or to obtain 4-way handshake required to bruteforce WPA.

Changing the 802.11 capture modes is very platform and driver dependent and Windows is very limited here. Monitor mode works with some Atheros chipset based cards with appropriate drivers but thats another story. Unless you don't have AirPcap – wireless packet capture solution for MS Windows environments this could be very painful so for this article we are going to use Linux operating system. Particularly BackTrack would be the vises choice as it has Wireshark and other tools pre-installed with the best wireless support available. Also try out TShark (command-line based network protocol analyzer), or Dumpcap (network traffic dump tool) for if you are not a GUI fan.

Packets Capture

Wireshark can capture traffic from many different network media types, including *wireless LAN* as well. Threats to wireless local area networks

(WLANs) are numerous and potentially devastating. In this article we will focus mostly on (undetected) wireless sniffing. Lets look at some simple examples how attacker may use Wireshark to compromise your infrastructure.

The process of wireless traffic sniffing can pose a number of challenges. In order to begin sniffing wireless traffic with Wireshark, your wireless card must be in monitor mode. Determine chipset/driver of your interface and check for monitor support mode or get supported one. This is not covered here. Wireshark does not do this automatically, you have to it manually. I suggest to use airmon-ng for all drivers except madwifi-ng to put your card into monitor mode. This script can be used to enable monitor mode on wireless interfaces. It may also be used to go back from monitor mode to managed mode. Entering the airmon-ng command without parameters will show the interfaces status.

```
Usage: airmon-ng <start|stop> <interface> [channel]
```

For never chipsets there is airmon-zc script which is intended to replace airmon-ng in 1.3 and is functionally based on it. Selecting a static channel is recommended in order to avoid packet loose.

```
root@bt:~# airmon-ng start wlan0 4
Interface Chipset Driver
wlan0 Atheros AR5414 ath5k - [phy0]
(monitor mode enabled on mon0)
```

To confirm that the card is in monitor mode, run the iwconfig command or rerun airmon-ng without any parameters. If you see output similar like above the wireless card is operating in monitor mode.

Fire up Wireshark, examine the detailed capture options if needed, choose your interface and start packet capture: Figure 1.

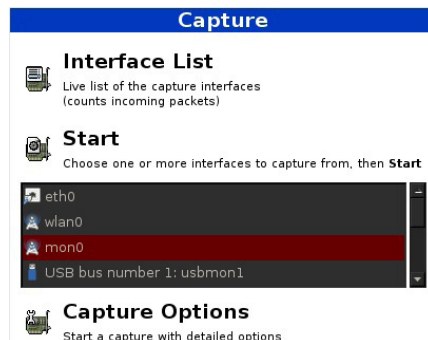


Figure 1. *Capture-interface*

Please ensure that you are capturing packets that belong to your network only!

Inspecting Packets

Click a packet to select it and you can dig down to view its details. The top panel is where captured data packets are listed, and they are usually ordered by the time they were sent. Underneath the Packet List (the second of the three panels) is the Packet Details window. This shows the data contained within the packet of data selected in the packet list. The third and final panel is the Packet Bytes panel. This panel reveals all the data that was sent or received as hexadecimal binary. There is also intuitive statistics menu available to display all kind of summaries, graphs allows user to sort packets.

Display filters

First time user may be surprised of “packet storms” flying around Wireshark, but there is nothing to be afraid of. This is the place when display filters can be handy. Display filters are used to change the view of a capture file. Before, when observing detailed capture options, you may noticed capture filter option. The main difference between capture filters and display filters is capture filter must be set before launching the Wireshark capture. Display filter can be modified at any time. Wireshark allows live capture and offline analysis of hundreds of protocols combined with powerful display filters. Display filters allows to display only selected packets by protocol, frame types, fields, values... When using a display filter, all packets remain in the capture file. The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type “dns” and you’ll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter. You can also click the Analyze menu and select Display Filters to create a new filter.

Extensive explanation and list of display filters is beyond of scope of this article, so few examples only:

- encryption mechanism is used to encrypt the contents of the frame:

```
wlan.fc.protected
```

- identify all unencrypted wireless traffic:

```
wlan.fc.protected ne 1
```

- BSSID filter, exclude traffic from any other APs:

```
wlan.bssid eq 00:11:22:33:44:55
```

- identify hidden SSID:

```
wlan.bssid eq 00:11:22:33:44:55 and wlan.fc.type_subtype eq 0
```

Building a custom filter is very easy. Build some filter and save them for future use. Lets say we want to see only DNS traffic comes from one single IP address and all we care about is our wireless access point. Filter would looks like this:

```
dns && wlan.bssid eq 00:11:22:33:44:55 && ip.src == 192.168.2.102
```

or all we care about is HTTP traffic contains plaintext “admin”:

```
http contains “admin”
```

Detecting Wireless Attack

Wireshark isn’t an intrusion detection system, however, it can be used as such. One of the most interesting purposes for network security engineers is its ability to use it to examine security problems. Networks using 802.1.1 are also subject to a number of *denial of service* (DoS) attacks that can render a WLAN inoperable. Network administrator suspects there is something wrong around wireless network. He applies filter for Deauthentication frame subtype and examine the content (Figure 2).

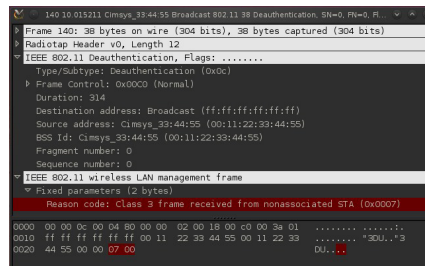


Figure 2. Wireshark-deauth-attack

As you can see there is ongoing aireplay-ng deauth attack (deauthenticate 1 or all stations (-0)). This filter can be also used to detect all kind of attack causing denial of service (MDK3).

Useful filter strings:

```
wlan.fc.type == 0 Management frames
wlan.fc.type == 1 Control frames
wlan.fc.type == 2 Data frames
wlan.fc.type_subtype == 0 Association request
wlan.fc.type_subtype == 1 Association response
wlan.fc.type_subtype == 2 Reassociation request
wlan.fc.type_subtype == 3 Reassociation response
wlan.fc.type_subtype == 4 Probe request
wlan.fc.type_subtype == 5 Probe response
```

```
wlan.fc.type_subtype == 8 Beacon
```

Sniffing Unencrypted Traffic

By default, wireless routers and access points have security turned off. Wireshark passively captures packets and allows us to examine their content. In a WLAN environment, this protection is no longer enough since a wireless network can be accessed remotely from a distance without the need for a physical connection anyone using compatible wireless equipment can potentially access the LAN. Networks that use wireless are vulnerable whether they are switched or not. When there is no encryption at all – public Hot spots, you never know who is listening. When surfing the websites using normal HTTP protocol / data sent over port 80 will be in plain text so without even knowing anything about network protocols, even script kiddie can view the unencrypted data contained within each packet clearly. The technique of finding a password with Wireshark is relatively simple. Coloring rules can be applied to the packet list for quick, intuitive analysis. There are protocol decoders (or dissectors, as they are known in Wireshark) for a great many protocols. Different packets are shown in different colors in the packet lists. For start, we are going to use simple “http filter” to see only HTTP packets no matter from what source it comes from. There is very useful mechanism available in Wireshark for packet colorization. By default HTTP packets are colored green, but you can change that in Coloring Rules under the View menu if needed. Lets assume that your wireless router does not support secure login, turn off encryption of your wireless router, and try to log in into web interface using another wireless interface. You will see many packets flying around, apply http filter and hit CTRL+F to find the right packet contains your password entered before. Mark string to be found in packet details and see how easy this was (Figure 3).

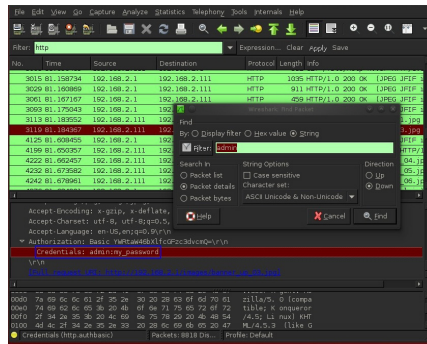


Figure 3. Wireshark-http-pass-sniff

Sniffing Encrypted Traffic

In order to start wireless sniffing we have to decrypt the traffic. Wireshark is armed with decryption support for many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2. The 802.11 dissector supports WEP and WPA/WPA2 decryption. In order to decrypt traffic, attacker should use other security tools and computing power to obtain credentials. There is nothing unusual to find hidden SSID in matter of seconds, crack WEP key in less than ten minutes but... Let me use well known saying I see every day when booting my favorite Linux operating system “The quieter you become, the more you can hear”. More recently, IDS have been developed for use on wireless networks. These wireless IDS can monitor and analyze user and system activities, recognize patterns of known attacks, identify abnormal network activity, and detect policy violations for WLANs. To reduce the risk of capture, hackers use passive OS fingerprinting on their target. Sniffers identify the operating systems on a network by the type of traffic they send and how they respond to traffic they receive. Patient attacker will sniff your traffic passively and gather all information about network infrastructure, not to risk to be uncovered by Intrusion Detection Systems / Wireless Intrusion Detection Systems. Wireless intrusion detection systems can identify even packet injection attack and warn the administrator.

Many companies have firewalls, intrusion detection systems, a solid authentication methods, strict password politics and all kind of security mechanism in place but there is always weak point somewhere. I have seen so many meeting rooms inside companies complex with no encryption at all because comfort is what matters. It would be not that hard to rent a near flat, use directional antenna and sniff all the traffic around. If there is some

network activity it shouldn't take more than few hours to collect enough initialization vectors to crack WEP key.

Adding Keys: 802.11 Preferences

Once entered (Edit/Preferences/Protocols/IEEE 802.11), there is no difference between sniffing unencrypted traffic and encrypted with Wired Equivalent Privacy security algorithm (Figure 4).

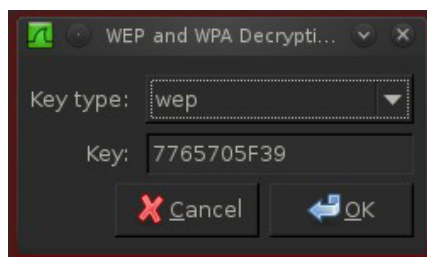


Figure 4. *Wireshark-decode-wep*

Decoding & Sniffing WPA

Cracking WPA is nowadays not that hard. Simple and often short passphrase makes this very easy for malicious attacker which often do have solid computing resources. Recently, faulty underlying design of the WPS PIN method on routers makes it easier for an attacker to crack the PIN combination by brute force using software tools that repeatedly guess the PIN. Depending on the exact wireless router, these tools can usually figure out a network's PIN and full WiFi password (the WPA or WPA2 passphrase) within a few hours. Don't forget that many routers have WiFi Protected Setup enabled by default. Assume this is the security whole attacker used to obtain WPA password. Just like before, enter WPA key into Wireshark preferences, but no traffic at all seems to be decoded? WPA and WPA2 use keys derived from an EAPOL handshake to encrypt traffic. Attacker would apply eapol filter and wait till client connects to access point or deauthenticate one or all stations to force them to reconnect (Figure 5).

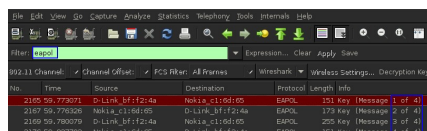


Figure 5. *Wireshark-eapol*

Theory says that unless all four handshake packets are present for the session we are trying to decrypt, Wireshark won't be able to decrypt the

traffic.

But it doesn't need message 3 for anything. Feel free to play with eapol filter and make your own conclusion.

FTP is one of the most commonly used means of transferring large amounts of data. After a while, attacker often observes the most valued IP address in the network. As you can see we have applied simple display filter to view only FTP packets from single host which is our point of interest and wireless access point we are sniffing. Another simple example of compromising FTP password being captured from the air (Figure 6).

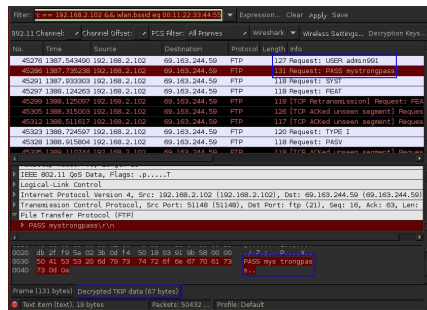


Figure 6. Wireshark-decrypted-tkip-sniffing-ftp-pass

Used Display Filter

```
ftp and ip.src == 192.168.2.102 && wlan.bssid eq 00:11:22:33:44:55
```

Our password has been compromised. See down left corner of screenshot, as as indicated, we gathered decrypted TKIP data along with 4-way handshake and decrypted FTP password successfully. You may also notice that this password is easily guessable so choosing strong one with special characters would be appropriate.

Following TCP Streams

One of the greatest analysis features is ability to view TCP streams as the application layer sees them. Rather than viewing data being send from client to server in a bunch of small chunks, the TCP stream feature sorts the data to make it easily viewable. One can spend a lot of time writing down the information from each packet and combining it to find out that is being said in the chat, but that is a bit time consuming and not really practical. Useful things to do is right click on a packet of interest and select "Follow TCP Stream" option this will give you the transactions that happened between two points, perfect for reassembling an AIM conversation. We

could go further with capturing and decoding SIP/VoIP traffic but previous demonstrations should be enough.

Facebook – the place for social engineering attacks may reveal sensitive informations that can be later used. We still have our wireless interface in monitor mode and we are able to decrypt WPA-TKIP but not when comes to secure connection. Facebook has added a new feature to browse the popular social network on a secure connection. However, it is not yet turned on by default. So the recommendation is to always use HTTPS or you have no privacy at all. After a while, when searching for plain text around HTTP packets there is a message sniffed from chat... (Figure 7).

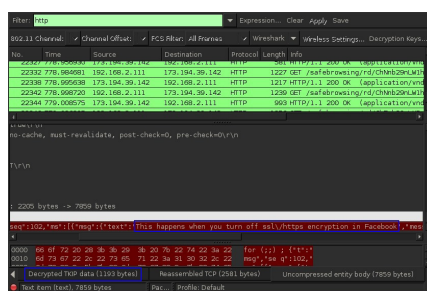


Figure 7. Wireshark-sniffing-facebook-chat

When there is “some” encryption present, setting rogue access point should do the trick too. Wireshark can decrypt SSL traffic as long as you have the private key, but the question if the key is really necessary. The rogue AP can be configured to looks like a legitimate AP and, since many wireless clients simply connect to the AP with the best signal strength, users can be “tricked” into inadvertently associating with the rogue AP. Tools like Airbase-ng will eventually convict victim access point to choose... Once a user is associated, all communications can be monitored by the hacker through the rogue AP.

Now is the time for previously mentioned promiscuous mode. Promiscuous mode allows a network device to intercept and read each network packet that arrives in its entirety. This mode is normally used for packet sniffing that takes place on a router or on a computer connected to a hub (instead of a switch) or one being part of a WLAN.

At this stage attackers are not longer worried about IDS or other security mechanisms because all malicious attempts runs outside protected network. Once they have accessed systems, intruders can launch denial of service attacks, steal identities, violate the privacy of

legitimate users, insert viruses or malicious code, and disable operations. Common man in the middle attack, exploit kits takes their places from here and takes care even about SSL.

One simple note – if there is an access point in range with SSID same or similar to company's name it not always have to be access point under company's control. Once an unauthorized user has gained access to the network, monitoring of the now unprotected data can lead to user names and passwords being intercepted, which can then be used for further attacks like stealing authentication cookies.

If this short article encourages you get your hands on Wireshark, don't hesitate and get your shark now from wireshark.org Take your time and study well written documentation which will take you step by step through wonderful experiences.

Conclusion

WLAN devices based on the IEEE 802.11 standard have a number of vulnerabilities related to the fact that wireless signals are sent over the air rather than through closed wiring paths. In WLANs, network traffic is broadcast into uncontrolled public spaces, which may result in the compromise of sensitive information. Always use the highest security methods of encryption possible and lower AP transmit power. Security is a process, not an instant soup. Discovering one even simple vulnerability could lead to compromise whole network.

MI1

MI1 is a security enthusiast with university degree in the field of informatics currently working for one of Europe's largest IT and Telecommunications service provider. He is the founder of hack4fun.eu where you can reach his thoughts written in English or Slovak language.

Using Wireshark and Other Tools to as an Aid in Cyberwarfare and Cybercrime

Attempting to Solve the “Attribution Problem” – Using Wireshark and Other Tools to as an Aid in Cyberwarfare and Cybercrime for

Analyzing the Nature and Characteristics of a Tactical or Strategic Offensive Cyberweapon and Hacking Attacks.

One of the main disadvantages of the hyper-connected world of the 21st century is the very real danger that countries, organizations, and people who use networks computer resources connected to the Internet face because they are at risk of cyberattacks that could result in anything ranging from denial service, to espionage, theft of confidential data, destruction of data, and/or destruction of systems and services. As a recognition of these dangers, the national leaders and military of most modern countries have now recognized that the potential and likely eventuality of cyberwar is very real and many are preparing to counter the threats of cyberwar with modern technological tools using strategies and tactics under a framework of cyberdeterrence, with which they can deter the potential attacks associated with cyberwarfare.

What is Cyberwarfare?

During my studies prior to and as a student in this DET 630 – Cyberwarfare and Cyberdeterrence course at Bellevue University, it occurred to me that considering the rapid evolution of the potentially destructive capabilities of cyberweapons and the complex nature of cyberdeterrence in the 21st century, it is now a critical priority to integrate the cyberwarfare and cyberdeterrence plans into the CONOPS plan. Indeed, if the strategic battleground of the 21st century has now expanded to include cyberspace, and the U.S. has in the last five years ramped up major military commands, training, personnel, and capabilities to support cyberwarfare and cyberdeterrence capabilities, the inclusion of these capabilities should now be a critical priority of the Obama administration if has not already happened.

How large a problem is this for the United States?

Without the integration of cyberwarfare and cyberdeterrence technologies, strategies, and tactics into the CONOPS Plan, the national command authorities run a grave risk of conducting a poorly planned offensive cyberwarfare operation that could precipitate a global crisis, impair relationships with its allies, and potentially unleash a whole host of unintended negative and potentially catastrophic consequences. In non-

military terms, at least four notable cyberspace events caused widespread damages via the Internet because of the rapid speed of their propagation, and their apparently ruthless and indiscriminant selection of vulnerable targets. They are 1) the Robert Morris worm (U.S. origin, 1988); 2) the ILOVEYOU worm (Philippines origin, 2000); the Code Red worm (U.S. origin, 2001); and the SQL Slammer worm (U.S. origin, 2003). If not executed with great care and forethought, a cyberweapons could potentially unleash even greater damage on intended targets and possible on unintended targets that were connected via the Internet.

Other Not So Obvious Challenges for Cyberweapons and Cyberdeterrence

The cyberspace threat and vulnerability landscape is notable in that it is continually dynamic and shifting. Those who are responsible for protecting assets in cyberspace have many more challenges on their hands than their military counterparts who utilize weapons like guns, explosives, artillery, missiles, etc. For example, there are by some estimates over 350 new types of malware that are manufactured each month. There are also monthly patch updates to most Microsoft software and operating systems, and phenomena such as evil hackers and zero-day exploits are apparently never ending. Therefore, the inclusion of cyberweapons and cyberdeterrence capabilities into the CONOPS Plan would require more frequent, rigorous, complex, and integrated testing to ensure that it was always effective and up to date. In the dynamic world of cyberspace with its constantly shifting landscape of new capabilities, threats and vulnerabilities, the coordination of the constant refresh and testing of a CONOPS Plan that integrated these cyberwarfare and cyberdeterrence capabilities would be no small feat. In addition, constant intelligence gathering and reconnaissance would need to be performed on suspected enemies to ensure that our cyberweapons and cyberdeterrence capabilities would be in constant state of being able to deliver the intended effects for which they were designed.

Is it a problem for other countries?

The careful planning and integration of cyberweapons and cyberdeterrence is likely a challenge for every country with these capabilities. For example, much is already known about our potential adversaries, such as Russia, China and North Korea, but what is perhaps less understood is the degree to

which they have been successful in integrating cyberwarfare and cyberdeterrence capabilities into their own national war plans. Nevertheless, due to the previous extensive experience of Russia and the U.S. with strategic war planning, it is more likely that each of these countries stand the greatest chance of making integrating cyberwarfare and cyberdeterrence capabilities into their respective war plans. Yet, as far back as June 2009, it was clear that the U.S. and Russia were unable to agree on a treaty that would create the terms under which cyberwarfare operations could and would be conducted (Markoff, J. and Kramer, A. E., 2009).

Is it problematic for these countries in the same ways or is there variation? What kind?

Every country that is modern enough to have organizations, people, and assets that are connected to computers and the Internet faces similar challenges of planning and managing cyberweapons and cyberdeterrence, and the poorer the country, the more significant the challenges. For example, when a small group of hackers from Manila in the Philippines unleashed the ILOVEYOU worm on the Internet in 2000, it caused over \$2 billion in damages to computer data throughout the world. Agents from the FBI went to Manila to track down these people and investigate how and why the ILOVEYOU worm catastrophe occurred. To their surprise, they learned that each of these hackers who were involved could successfully escape prosecution because there were no laws in the Philippines with which to prosecute them. So actually most countries lack the technological and legal frameworks with which to successfully build a coordinated effort to manage the weapons and strategies of cyberwarfare and cyberdeterrence, despite the fact that most now embrace cyberspace with all the positive economic benefits it offers for commerce and communications.

What are the consequences to the U.S. and others if this threat is left unchecked?

As stated earlier, without the careful integration of cyberwarfare and cyberdeterrence technologies, strategies, and tactics into the CONOPS Plan, the national command authorities run a grave risk of launching a poorly planned offensive cyberwarfare operation that could precipitate a global crisis, impair relationships with its allies, and potentially unleash a whole host of unintended negative and potentially catastrophic consequences.

What consequences has the threat already produced on American/global society?

I believe that yes, the absence of well-defined cyberwarfare and cyberdeterrence strategies and tactics in the CONOPS Plan has already produced some situations that have either damaged America's image abroad, or that could imperil its image and have far more negative consequences. For example, operations such as Stuxnet, Flame, Duqu, etc., might have either been better planned or possibly not executed at all if cyberwarfare and cyberdeterrence strategies and tactics were defined in the CONOPS Plan. Also, the news media indicated during the revolution in Libya that resulted in the fall of Qaddafi, cyberwarfare operations were considered by the Obama administration. The negative reactions and repercussions on the world stage might have far outweighed any short term advantages that could have resulted from a successful set of cyberattacks against Libyan infrastructure assets that were attached to computer networks. Again, a comprehensive CONOPS Plan that included well-defined cyberwarfare and cyberdeterrence strategies and tactics could have prevented such possible cyberattacks from even being considered, and it could have prevented the news of the possible consideration being publicized in the press (Schmitt, E. and Shanker, T., 2011). Without such restraint and well-planned deliberate actions, the U.S. runs the risk of appearing like the well-equipped cyber bully on the world stage, and an adversary who is willing to unleash weapons that can and will do crippling damage to an opponent, using technologies that are rapid, decisive, and not well-understood by those for whom they are intended. A similar effect and world reaction might be if U.S. Army infantry troops were equipped with laser rifles that emitted deadly laser blasts with pinpoint precision across several hundred yards.

Has this threat evolved or changed over time or is it relatively constant? If it has evolved or changed, exactly how has that change happened and what political consequences have emerged from them?

The threat has certainly rapidly evolved over time. Since Stuxnet was released in 2010, countries and the general public are now aware of some of

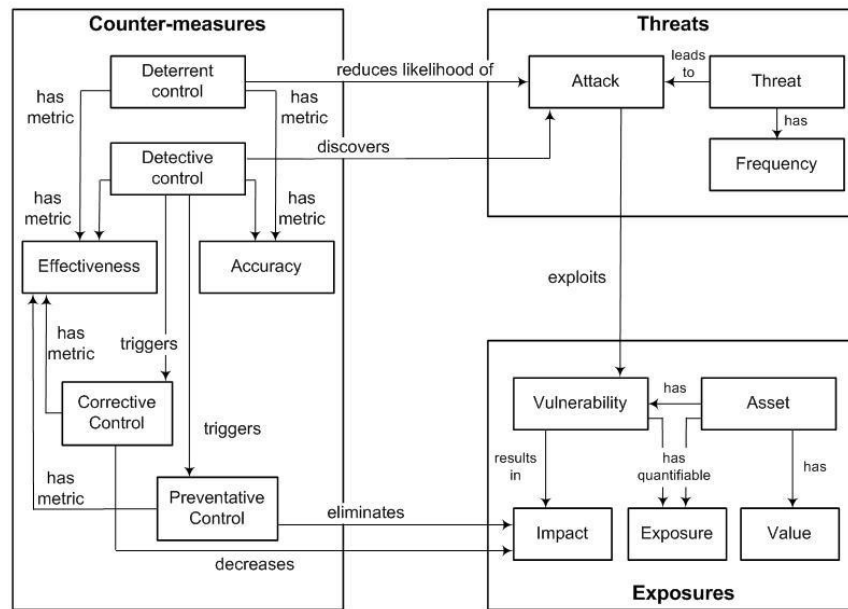
the offensive, strategic and destructive capabilities and potential of cyberweapons (Gelton, T., 2011).

The changes that produced Stuxnet and other recent, more modern cyberweapons were a national resolve to excel in the cyberwarfare area, coupled with excellent reconnaissance on desired targets, and partnering with computer scientists in Israel. The political consequences are not well understood yet, except to say that the U.S. and Israel are probably less trusted and suspected of even greater future capabilities, as well as having the will to use them. Again, having well-planned cyberwarfare and cyberdeterrence strategies and tactics defined in the CONOPS Plan might indeed, restrain such possibly reckless decisions as to unleash cyberweapon attacks without what the world might consider the correct provocation.

Final Thoughts about Cyberwarfare Operations

In the words of Deb Radcliff, in an article published in SC Magazine in September 2012, “we are already in a cyberwar” (Radcliff, D., 2012). But as I was performing my research, it occurred to me that a country like the U.S., might in the future unleash such a devastating cyberattack that it could cripple the enemy’s ability to communicate surrender. I think that the moral implications of such circumstances need to be justly considered as a matter of the laws of war, because if a country continues to attack an enemy that has indicated that they are defeated and want to surrender, this shifts the moral ground from which the U.S. may have it was conducting its cyberwarfare operations. This is one other unintended consequence of cyberwarfare and one that needs to be carefully considered.

To further understand the relationship of threats, counter-measures, and exposures in cyberspace, I have included this diagram by Jaquith, shown Figure 1.



Logical Model of IT Security Management Controls
 From Security Metrics by Andrew Jaquith, published by Addison-Wesley, 2007

Figure 1. Logical Model of IT Security Management Controls (Jacquith, 2007)

The Attribution Problem

One of the most perplexing issues of cyberwarfare and cybercrime is the fact that attackers can and very often will use software and other servers from which to launch their attacks. Because of the way the Internet was designed its end-to-end nature of IP communications using other computers to launch attacks is not that difficult. In fact, the computers that actually perform the attacks are called “zombies” as they are configured with remote control programs that are manipulated by the attackers. The recipients can do forensic analysis and determine which “zombie” computers sent the attacks, however, it is practically impossible to collect the data about who the person or persons that originated the attacks. Thus, it is very difficult to attribute the original cause of the attack, hence the name the “attribution problem.” In cyberwarfare, this is particularly difficult, because the National Command Authorities would want to understand to whom and where they should employ the cyberwarfare capable units of the U.S. Military to launch a punishing retaliatory cyberattack.

The most common type of attack for “zombie” computers is known as the distributed denial of service attack or DDoS attack. In February 2000, the

first sensational wave of DDoS attacks were launched from “zombie” computers that were physically located at major universities in California. The following figures provide some of the details about those attacks and which companies were the targets (Figure 2-4).



Figure 2. Denial of Service Attack diagram from ABC news in February 2000



Figure 3. Denial of Service Attack Victims diagram from ABC news in February 2000



Figure 4. Denial of Service Attack Zombies diagram from ABC news in February 2000

Recent Cyber Attacks

As recently as September 23, 2012 – September 30, 2012, cyber attacks in the form of distributed *denial of service* (DDOS) attacks from the Middle East against several major U.S. banks based have publicly demonstrated the ire of the attackers and also the vulnerabilities of banks with a customer presence in cyberspace (Strohm and Engleman, 2012).

How do you know?

It's not always intuitively obvious, but if your network is slowing down or computers or other devices attached to your network are acting strangely,

you could be under attack. But it's best to use analysis tools to understand what is really going on.

Free Tools You Can Use

This section covers three free tools that you can use to understand network activity on your network in greater detail.

Wireshark

Wireshark is a free, open source packet analysis tool that evolved from its predecessor, Ethereal.

Wireshark is notable for its ability to quickly, capture and display traffic in a real time sequential way, and allow this traffic to be displayed, broken down at the packet level by each level of the OSI model, from the physical layer up through the application layer. The traffic can also shows the senders and the receivers of each packet, and can be easily summarized with the selection of a few menu choices. The first figure below is from a table in the Wireshark documentation, and the figures that follow are from an actual Wireshark session where about 500,000 packets were collected for summarization and analysis. All this data can also be saved for later analysis.

Wireshark will run on both Windows-based platforms and Mac OS X platforms. This is the website location where you can find Wireshark: <http://www.wireshark.org/download.html> (Table 1 and Figure 5-8).

Table 1. *Wireshark Documentation – Packet Analysis Capabilities for Captured Packets*
The menu items of the "Packet List" pop-up menu

Item	Identical to main menu's item:	Description
Mark Packet (toggle)	Edit	Mark/unmark a packet.
Ignore Packet (toggle)	Edit	Ignore or inspect this packet while dissecting the capture file.
Set Time Reference (toggle)	Edit	Set/reset a time reference.
Manually Resolve Address		Allows you to enter a name to resolve for the selected address.
Apply as Filter	Analyze	Prepare and apply a display filter based on the currently selected item.
Prepare a Filter	Analyze	Prepare a display filter based on the currently selected item.

Conversation - Filler	-	This menu item applies a display filter with the address information from only selected packet. E.g. the IP menu item will filter to show the traffic between the two IP addresses of the current packet. XXX - add a new section describing this better.
Cobrize Conversation	-	This menu item uses a display filter with the address information from the selected packet to build a new coloring rule.
SCTP	-	Allows you to analyze and prepare a filter for this SCTP association.
Follow TCP Stream	Analyze	Allows you to view all the data on a TCP stream between a pair of nodes.
Follow UDP Stream	Analyze	Allows you to view all the data on a UDP data stream between a pair of nodes.
Follow SSL Stream	Analyze	Same as "Follow TCP Stream" but for SSL. XXX - add a new section describing this better.

Copy/ Summary (Text)	-	Copy the summary fields as displayed to the clipboard, as tab-separated text.
Copy/ Summary (CSV)	-	Copy the summary fields as displayed to the clipboard, as comma-separated text.
Copy/ As Filter	-	Prepare a display filter based on the currently selected item and copy that filter to the clipboard.
Copy/ Bytes (Offset Hex)	-	Copy the packet bytes to the clipboard in hexdump-like format, but without the text portion.
Copy/ Bytes (Printable Text Only)	-	Copy the packet bytes to the clipboard as ASCII text, excluding non-printable characters.
Copy/ Bytes (Hex Stream)	-	Copy the packet bytes to the clipboard as an unquoted list of hex digits.
Copy/ Bytes (Binary Stream)	-	Copy the packet bytes to the clipboard as raw binary. The data is stored in the clipboard as MIME-type "application/octet-stream".

Decode As...	Analyze	Change or apply a new relationship between two dissectors.
Print...	File	Print packets.
Show Packet in New Window	View	Display the selected packet in a new window.

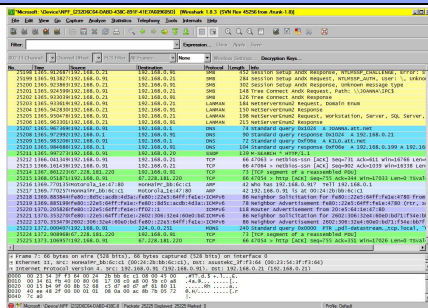


Figure 5. Wireshark Opening Screenshot after a Network Interface Has Been Selected for Packet Capture

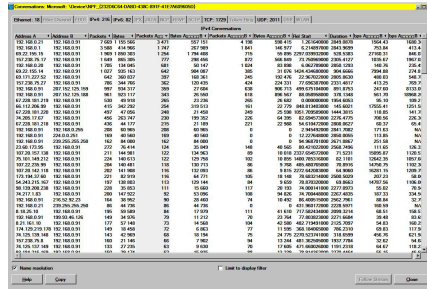


Figure 6. Wireshark Conversation Analysis Screen

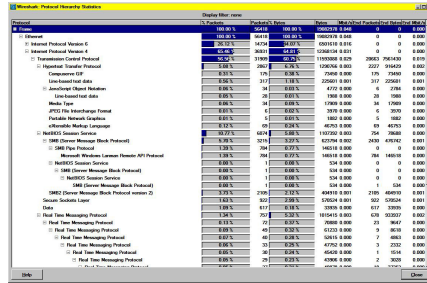


Figure 7. Wireshark Protocol Analysis Screen

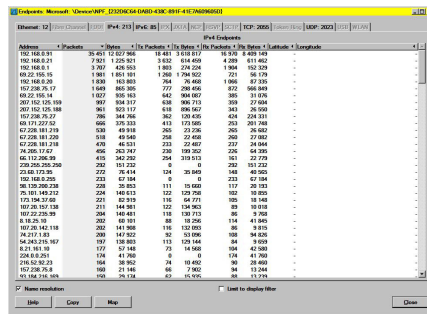


Figure 8. Wireshark Endpoint Analysis Screen

Ostinato

Ostinato is a free, open source-based packet generator that can be used to conduct network experiments, particularly for packet analysis in conjunction with a tool such as Wireshark. It is easy to install, configure and use. Figure 8 shows a screenshot from Ostinato.

Ostinato will run on Windows-based platforms and several other platforms.

This is the website location where you can find Ostinato:

<http://code.google.com/p/ostinato/> (Figure 9).

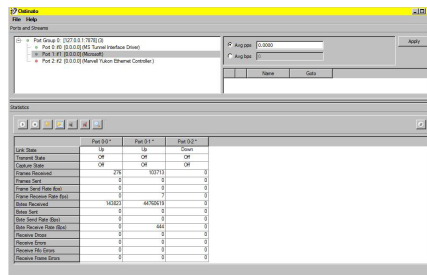


Figure 9. Ostinato Packet Generator Screen

TCPView

TCPView is an excellent analysis program that shows what is happening on your computer at layer four of the OSI networking model. If you remember, this is where TCP and UDP activities take place. TCPView allows the user to view and sort data by process, PID, protocol (TCP or UDP), local address, remote address, port number, TCP state, sent packets, sent bytes, received packets, and received bytes. The data can also be saved for later analysis.

TCPView was originally written by Mark Russinovich and Bryce Cogswell and was published and distributed for free by their company, Sysinternals. In 2006, Microsoft acquired Sysinternals and TCPView and many other tools that were created by Sysinternals continue to be updated and distributed

by Microsoft for free. TCPView will only run on Windows-based platforms and this is the website location where you can find TCPView and many other great Sysinternals tools: <http://technet.microsoft.com/en-us/sysinternals> (Figure 10).

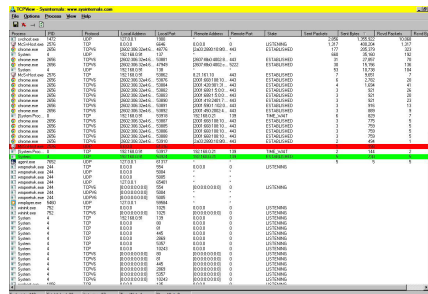


Figure 10. TCPView in operation, with records sorted by sent packets, in descending order

Traffic to Watch

By far the most interesting and dangerous external traffic to watch on most networks is ICMP traffic. ICMP is the Internet Control Messaging Protocol, and there are eight types of ICMP messages. Hackers can easily use ICMP (PING) messages to create DDOS attacks. A tool like Simple Nomad's "icmpenum" can issue ICMP messages such as ICMP_TIMESTAMP_REQUEST and ICMP_INFO and make it possible to map a network inside of a firewall (K, 2011).

Outbound traffic is just as important as inbound traffic if not more so (Geers, 2011). It is not uncommon for programs like botnets to take up residence and open up secure channels to transmit data to remote servers in places like China, Russia, Eastern Europe and even North Korea. Programs that are unrecognizable should be suspected as possible malware and should be quickly researched to determine if they are hostile. If they cannot be easily identified, that is a bad sign and they should probably be uninstalled.

A Caution to those Who Understand Network Attacks

Title 10 of the U.S. Code forbids U.S. Citizens from taking offensive action against network attackers. Nevertheless, monitoring the evidence and results of unwanted traffic could help you understand it and also help you decide how to improve upon your network defenses (firewall settings for inbound traffic, desktop firewalls, etc.) and even provide evidence to law enforcement authorities.

The Future

Without trying to present a gloomy picture of the cyberspace environment that is composed of the Internet and all the computers, smart phones and other devices attached to it, it appears that for the time being, the bad guys far outnumber the good guys and it appears that they are winning. But it is also apparent that that now more free information and free tools are available than ever before. For the foreseeable future, every person who uses the Internet should seek to educate themselves about the dangers in cyberspace and the ways to protect themselves from these dangers.

Conclusion

This article has briefly reviewed the topic of cyberwarfare and presented some information about free network analysis tools that can help you better understand your network traffic.

The good news is that President Obama and his Administration have an acute awareness of the importance of the cyberspace to the American economy and the American military. The bad news is that because we are already in some form of cyberwarfare that appears to be rapidly escalating,

it remains to be seen what effects these cyberattacks and the expected forthcoming Executive Orders that address cybersecurity will have on the American people and our way of life. I believe it will be necessary to act prudently, carefully balancing our freedoms with our need for security, and also considering the importance of enabling and protecting the prosperity of the now electronically connected, free enterprise economy that makes the U.S. the envy of and the model for the rest of the world.

References

- Andreasson, K. (Ed.). (2012). *Cybersecurity: Public Sector Threats and Responses*. Boca Raton, FL: CRC Press.
- Andress, J. and Winterfeld, S. (2011). *Cyber Warfare: Techniques and Tools for Security Practitioners*. Boston, MA: Syngress.
- Andreasson, K. (ed.). (2012). *Cybersecurity: Public Sector Threats and Responses*. Boca Raton, FL: CRC Press.
- Barnett, M. B. and Finnemore, M. (2004). *Rules for the World: International Organizations in Global Politics*. Ithaca, NY: Cornell University Press.
- Bayles, A., et al. (2007). *Penetration Tester's Open Source Toolkit, Volume 2*. Burlington, MA: Syngress.
- Blitz, A. (2011). *Lab Manual for Guide to Computer Forensics and Investigations, fourth edition*. Boston, MA: Course Technology, Cengage Learning.
- Bousquet, A. (2009). *The Scientific Way of Warfare: Order and Chaos on the Battlefields of Modernity*. New York, NY: Columbia University Press.
- Brancik, K. (2008). *Insider Computer Fraud: An In-Depth Framework for Detecting and Defending Against Insider IT Attacks*. Boca Raton, FL: Auerbach Publications.
- Britz, M. T. (2009). *Computer Forensics and Cyber Crime: An Introduction, second edition*. Upper Saddle River, NJ: Prentice-Hall.
- Bush, G. W. (2008). *Comprehensive National Cybersecurity Initiative (CNCI)*. Published by the White House January 2008. Retrieved from <http://www.whitehouse.gov/cybersecurity/comprehensive-national-cybersecurity-initiative> on January 5, 2012.
- Calder, A. and Watkins, S. (2010). *IT Governance: A Manager's Guide to Data Security and ISO27001/ISO27002, 4th edition*. London, UK: Kogan Page.
- Carr, J. (2012). *Inside Cyber Warfare, second edition*. Sebastopol, CA: O'Reilly.
- Carrier, B. (2005). *File System Forensic Analysis*. Upper Saddle River, NJ: Addison-Wesley.
- Carvey, H. (2009). *Windows Forensic Analysis DVD Toolkit, second edition*. Burlington, MA:
- Casey, E. (2011). *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet, third edition*. New York, NY: Elsevier.
- Chappell, L. (2010). *Wireshark Network Analysis: The Official Wireshark Certified Network Analyst Study Guide, first edition*. San Jose, CA: Chappell University.
- Cialdini, R. B. (2009). *Influence: Science and Practice, fifth edition*. Boston, MA: Pearson Education.
- Clarke, R. A. and Knake, R. K. (2010). *Cyberwar: the Next Threat to National Security and What to Do About It*. New York, NY: HarperCollins Publishers.
- CNBC. (2012) *Cyber Espionage: The Chinese Threat. A collection of articles about the cyber threats posed by Chinese hackers*. Retrieved from <http://www.cnbc.com/id/47962207/> on July 10, 2012.
- Cole, E. and Ring, S. (2006). *Insider Threat: Protecting the Enterprise from Sabotage, Spying, and Present Employees and Contractors from Stealing Corporate Data*. Rockland, MA: Syngress Publishing, Inc.
- Cole, E., et al. (2009). *Network Security Bible, second edition*. Indianapolis, IN: Wiley Publishing, Inc.
- Czosseck, C. and Geers, K. (2009). *The Virtual battlefield: Perspectives on Cyber Warfare*. Washington, DC: IOS Press.
- Davidoff, S. and Ham, J. (2012). *Network Forensics: Tracking Hackers Through Cyberspace*. Upper Saddle River, NJ: Prentice-Hall.
- Dhanjani, N. (2009). *Hacking: The Next Generation*. Sebastopol, CA: O'Reilly.
- Edwards, M. and Stauffer, T. (2008). *Control System Security Assessments. A technical paper presented at the 2008 Automation Summit – A Users Conference, in Chicago*. Retrieved from the web at <http://www.infracritical.com/papers/nstb-2481.pdf> on December 20, 2011.
- Fayutkin, D. (2012). *The American and Russian Approaches to Cyber Challenges*. Defence Force Officer, Israel. Retrieved from <http://omicsgroup.org/journals/2167-0374/2167-0374-2-110.pdf> on September 30, 2012.
- Freedman, L. (2003). *The Evolution of Nuclear Strategy*. New York, NY: Palgrave Macmillan.
- Friedman, G. (2004). *America's Secret War: Inside the Hidden Worldwide Struggle Between America and Its Enemies*. New York, NY: Broadway Books.
- Geers, K. (2011). *Strategic Cyber Security. A Cybersecurity technical paper published at DEFCON 20*.
- Georgetown University. (2012). *International Engagement in Cyberspace part 1. A YouTube video*. Retrieved from <http://www.youtube.com/watch?v=R11FNgtui00&feature=related> on September 21, 2012.
- Gerwitz, D. (2011). *The Obama Cyberdoctrine: tweet softly, but carry a big stick*. An article published at Zdnet.com on May 17, 2011. Retrieved from <http://www.zdnet.com/blog/government/the-obama-cyberdoctrine-tweet-softly-but-carry-a-big-stick/10400> on September 25, 2012.
- Gjelten, T. (2010). *Are 'Stuxnet' Worm Attacks Cyberwarfare?* An article published at NPR.org on October 1, 2011. Retrieved from the web at <http://www.npr.org/2011/09/26/140789306/security-expert-u-s-leading-force-behind-stuxnet> on December 20, 2011.

- Gjelten, T. (2010). Stuxnet Computer Worm Has Vast Repercussions. An article published at NPR.org on October 1, 2011. Retrieved from the web at <http://www.npr.org/templates/story/story.php?storyId=130260413> on December 20, 2011.
- Gjelten, T. (2010). Stuxnet Computer Worm Has Vast Repercussions. An article published at NPR.org on October 1, 2011. Retrieved from the web at <http://www.npr.org/templates/story/story.php?storyId=130260413> on December 20, 2011.
- Gjelten, T. (2011). Security Expert: U.S. 'Leading Force' Behind Stuxnet. An article published at NPR.org on September 26, 2011. Retrieved from the web at <http://www.npr.org/2011/09/26/140789306/security-expert-u-s-leading-force-behind-stuxnet> on December 20, 2011.
- Gjelten, T. (2011). Stuxnet Raises 'Blowback' Risk In Cyberwar. An article published at NPR.org on December 11, 2011. Retrieved from the web at <http://www.npr.org/2011/11/02/141908180/stuxnet-raises-blowback-risk-in-cyberwar> on December 20, 2011.
- Gjelten, T. (2011). Stuxnet Raises 'Blowback' Risk In Cyberwar. An article published at NPR.org on December 11, 2011. Retrieved from the web at <http://www.npr.org/2011/11/02/141908180/stuxnet-raises-blowback-risk-in-cyberwar> on December 20, 2011.
- Glenny, M. (2011). Dark Market: Cyberthieves, Cybercops and You. New York, NY: Alfred A. Knopf.
- Grabo, C. M. (2004). Anticipating Surprise: Analysis for Strategic Warning. Lanham, MD: University Press of America, Inc.
- Guerin, J. (2010). The Essential Guide to Workplace Investigations: How to Handle Employee Complaints & Problems. Berkeley, CA: Nolo.
- Guerin, J. (2010). The Essential Guide to Workplace Investigations: How to Handle Employee Complaints & Problems. Berkeley, CA: Nolo.
- Harper, A., et al. (2011). Gray Hat Hacking: The Ethical Hacker's Handbook, third edition. New York, NY: McGraw Hill.
- Hintzbergen, J., et al. (2010). Foundations of Information Security Based on ISO27001 and ISO27002, second edition. Amersfoort, NL: Van Haren Publishing.
- Honker's Union of China. (2012). Honker's Union of China website. Retrieved from <http://www.huc.me/> on September 21, 2012.
- Hyacinthe, B. P. (2009). Cyber Warriors at War: U.S. National Security Secrets & Fears Revealed. Bloomington, IN: Xlibris Corporation.
- Jones, K. J., et al. (2006). Real Digital Forensics: Computer Security and Incident Response. Upper Saddle River, NJ: Addison-Wesley.
- Jones, R. (2006). Internet Forensics: Using Digital Evidence to Solve Computer Crime. Cambridge, MA, CA: O'Reilly.
- K., Dr. (2011). Hacker's Handbook, fourth edition. London, U.K.: Carlton.
- Kaplan, F. (1983). The Wizards of Armagedden: The Untold Story of a Small Group of Men Who Have Devised the Plans and Shaped the Policies on How to Use the Bomb. Stanford, CA: Stanford University Press.
- Kerr, D. (2012). Senator urges Obama to issue 'cybersecurity' executive order. An article published at Cnet.com on September 24, 2012 Retrieved from http://news.cnet.com/8301-1009_3-57519484-83/senator-urges-obama-to-issue-cybersecurity-executive-order/ on September 26, 2012.
- Knapp, E D. (2011). Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems. Waltham, MA: Syngress, MA.
- Kramer, F. D. (ed.), et al. (2009). Cyberpower and National Security. Washington, DC: National Defense University.
- Landy, G. K. (2008). The IT/Digital Legal Companion: A Comprehensive Business Guide to Software, IT, Internet, Media, and IP Law. Burlington, MA: Syngress.
- Langer, R. (2010). Retrieved from the web at <http://www.langner.com/en/blog/page/6/> on December 20, 2011.
- Libicki, M.C. (2009). Cyberdeterrence and Cyberwar. Santa Monica, CA: Rand Corporation.
- Lockhart, A. (2007). Network Security Hacks: Tips & Tools for Protecting Your Privacy, second edition. Sebastopol, CA: O'Reilly.
- Logicalis. (2011). Seven Ways to Identify a Secure IT Environment. Published at IT Business Edge in 2011. Retrieved from <http://www.itbusinessedge.com/slideshows/show.aspx?c=92732&placement=bodycopy> in May 5, 2011.
- Long, J., et al. (2008). Google Hacking for Penetration testers, Volume 2. Burlington, MA: Syngress Publishing, Inc.
- Long, J., et al. (2008). No Tech Hacking: A Guide to Social Engineering, Dumpster Diving, and Shoulder Surfing. Burlington, MA: Syngress Publishing, Inc.
- Markoff, J. and Kramer, A. E. (2009). U.S. and Russia Differ on a Treaty for Cyberspace. An article published in the New York Times on June 28, 2009. Retrieved from <http://www.nytimes.com/2009/06/28/world/28cyber.html?pagewanted=all> on June 28, 2009.
- Mayday, M. (2012). Iran Attacks US Banks in Cyber War: Attacks target three major banks, using Muslim outrage as cover. An article published on September 22, 2012 at Poltix.Topix.com. Retrieved from <http://politix.topix.com/homepage/2214-iran-attacks-us-banks-in-cyber-war> on September 22, 2012.
- McBrie, J. M. (2007). THE BUSH DOCTRINE: SHIFTING POSITION AND CLOSING THE STANCE. A scholarly paper published by the USAWC STRATEGY RESEARCH PROJECT. Retrieved from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA423774> on September 30, 2012.
- Middleton, B. (2005). Cyber Crime Investigator's Field Guide, second edition. Boca Raton, FL: Auerbach Publications.
- Mitnick, K. and Simon, W. (2002). The Art of Deception: Controlling the Human Element Security. Indianapolis, IN: Wiley Publishing, Inc.
- Mitnick, K. and Simon, W. (2006). The Art of Intrusion: The Real Stories Behind the Exploits of Hackers, Intruders & Deceivers. Indianapolis, IN: Wiley Publishing, Inc.
- Nelson, B., Et al. (2010). Guide to Computer Forensics and Investigations, fourth edition. Boston, MA: Course Technology, Cengage Learning.
- Northcutt, S. and Novak, J. (2003). Network Intrusion, third edition. Indianapolis, IN: New Riders.
- Obama, B. H. (2012). Defense Strategic Guidance 2012 – Sustaining Global Leadership: Priorities for 21st Century Defense. Published January 3, 2012. Retrieved from http://www.defense.gov/news/Defense_Strategic_Guidance.pdf on January 5, 2012.
- Obama, B.H. (2011). INTERNATIONAL STRATEGY for Cyberspace. Published by the White House on May 16, 2011. Retrieved from http://www.whitehouse.gov/sites/default/files/rss_viewer/international_strategy_for_cyberspace.pdf on May 16,

- 2011.
- Osborne, M. (2006). *How to Cheat at Managing Information Security*. Rockland, MA: Syngress.
 - Parker, T., et al. (2004). *Cyber Adversary Characterization: Auditing the Hacker Mind*. Rockland, MA: Syngress Publishing, Inc.
 - Payne, K. B. (2001). *The Fallacies of Cold War Deterrence and a New Direction*. Lexington, KY: The University of Kentucky Press.
 - Philipp, A., et al. (2010). *Hacking Exposed Computer Forensics: Secrets and Solutions*, second edition. New York, NY: McGraw-Hill.
 - Pry, P. V. (1999). *War Scare: Russia and America on the Nuclear Brink*. Westport, CT: Praeger Publications.
 - Radcliff, D. (2012). *Cyber Cold War*. An article published in the SC Magazine, September 2012 issue.
 - Radcliff, D. (2012). *Cyber cold war: Espionage and warfare*. An article published in SC Magazine, September 4, 2012. Retrieved from <http://www.scmagazine.com/cyber-cold-war-espionage-and-warfare/article/254627/> on September 7, 2012.
 - Reynolds, G. W. (2012). *Ethics in Information Tehnology*, 4th edition. Boston, MA: Course Technology.
 - Reynolds, G. W. (2012). *Ethics in Information Tehnology*, 4th edition. Boston, MA: Course Technology.
 - Rogers, R., et al. (2008). *Nessus Network Auditing*, second edition. Burlington, MA: Syngress.
 - Rosenbaum, R. (2011). *How the End Begins: The Road to a Nuclear World War III*. New York, NY: Simon and Schuster.
 - RT. (2012). *Iran may launch pre-emptive strike on Israel, conflict could grow into WWII – senior commander*. An article published at RT.com on September 23, 2012. Retrieved from <http://rt.com/news/iran-strike-israel-world-war-803/> on September 24, 2012.
 - Sanger, D. E. (2012). *Confront and Coneal: Obama’s Secret Wars and Surprising Use of America Power*. New York, NY: Crown Publishers.
 - Schell, B. H., et al. (2002). *The Hacking of America: Who’s Doing It, Why, and How*. Westport, CT: Quorum Press.
 - Schlesinger, J. (2012). *Chinese Espionage on the Rise in US, Experts Warn*. An article published at CNBC.com on July 9, 2012. Retrieved from <http://www.cnbc.com/id/48099539> on July 10, 2012.
 - Schmidt, H. S. (2006). *Patrolling Cyberspace: Lessons Learned from Lifetime in Data Security*. N. Potomoc, MD: Larstan Publishing, Inc.
 - Schmitt, E. and Shanker, T. (2011). *U.S. Debated Cyberwarfare in Attack Plan on Libya*. An article published in the New York Times on October 17, 2011. Retrieved from <http://www.nytimes.com/2011/10/18/world/africa/cyber-warfare-against-libya-was-debated-by-us.html> on October 17, 2011.
 - Seagren, E. (2007). *Secure Your Network for Free: Using NMAP, Wireshark, SNORT, NESSUS, and MRTG*. Rockland, MA: Syngress.
 - Seagren, E. (2007). *Secure Your Network for Free: Using NMAP, Wireshark, SNORT, NESSUS, and MRTG*. Rockland, MA: Syngress.
 - SEM. (2011). *The Hacker’s Underground*. Retrieved from <http://serpentsembrace.wordpress.com/2011/05/17/the-hackers-underground/> on September 21, 2012.
 - Simpson, M. T., et al. (2011). *Hands-On Ethical Hacking and Network Defense*. Boston, MA: Course Technology.
 - Skpudis, E. and Liston, T. (2006). *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses*, second edition. Upper Saddle River, NJ: Prentice-Hall.
 - Soloman, M. G., et al. (2011). *Computer Forensics Jump Start*, second edition. Indianapolis, IN: Wiley Publishing, Inc.
 - Stallings, W. (2011). *Network Security Essentials: Applications and Standards*, fourth edition. Boston, MA: Prentice Hall.
 - Stiennon, R. (2010). *Surviving Cyber War*. Lanham, MA: Government Institutes.
 - Strohm, C. and Engleman, E. (2012). *Cyber Attacks on U.S. Banks Expose Vulnerabilities*. An article published at BusinessWeek.com on September 28, 2012 Retrieved from <http://www.businessweek.com/news/2012-09-27/cyber-attacks-on-u-dot-s-dot-banks-expose-computer-vulnerability> on September 30, 2012.
 - Technolytics. (2011). *Cyber Commander’s eHandbook: The Weaponry and Strategies of Digital Conflict*. Purchased and downloaded from Amazon.com on April 16, 2011.
 - The Hacker’s Underground. An article published at the Serpent’s Embrace blog. Retrieved from <http://serpentsembrace.wordpress.com/tag/honker-union-of-china/> on September 21, 2012.
 - Trost, R. (2010). *Praactical Intrusion Analysis: Prevention and Detection for the Twenty-First Century*. Boston, MA: Addison-Wesley.
 - Vacca, J. R. (2002). *Computer Forensics: Computer Crime Scene Investigation*. Hingham, MA: Charles River Media.
 - van Wyk, K. R. and Forno, R. (2001). *Incident Response*. Cambridge, MA, CA: OReilly.
 - Verizon. (2012). *The 2012 Verizon Data Breach Investigations Report*. Retrieved from http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf on September 17, 2012.
 - Verizon. (2012). *The 2012 Verizon Data Breach Investigations Report*. Retrieved from http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf on September 17, 2012.
 - Volonino, L. and Anzaldua, R. (2008). *Computer Forensics for Dummies*. Hoboken, NJ: Wiley Publishing, Inc.
 - Waters, G. (2008). *Australia and Cyber-Warfare*. Canberra, Australia: ANU E Press.
 - Whitman, M. E. and Mattord, H. J. (2007). *Principles of Incident Response & Disaster Recovery*. Boston, MA: Course Technology – Cengage Learning.
 - Wikipedia Commons. (2011). *Stuxnet Diagram*. Retrieved from the web at http://en.wikipedia.org/wiki/File:Step7_communicating_with_plc.svg on December 20, 2011.
 - Wiles, J., et al. (2007). *Low Techno Security’s Guide to Managing Risks: For IT Managers, Auditors, and Investigators*. Burlington, MA: Syngress Publishing, Inc.
 - Wiles, J., et al. (2012). *Low Tech Hacking: Street Smarts for Security Professionals*. Waltham, MA: Syngress Publishing, Inc.
 - Wilhelm, T. and Andress, J. (2011). *Ninja Hacking: Unconventional Penetration Testing Tactics and Techniques*. Burlington, MA: Syngress Publishing, Inc.
 - Zalewski, M. (2005). *Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks*. San Francisco, CA: No Starch Press.
 - Zetter, K. (2011). *How Digital Detectives Deciphered Stuxnet, the Most Menacing Malware in History*. An article published on July 11, 2011 at Wired.com. Retrieved from the web at <http://www.wired.com/threatlevel/2011/07/how-digital-detectives->

[deciphered-stuxnet/all/1](#) on December 20, 2011.

- Zittrain, J. (2012). Professor Zittrain Q&A Hactivism: Anonymous, lulzsec, and Cybercrime in 2012 and Beyond. A YouTube video. Retrieved from <http://www.youtube.com/watch?v=CZWjfxY8nmU&feature=related> on September 21, 2012.

William F. Slater III

*William F. Slater, III, MBA, M.S., PMP, CISSP, SSCP, CISA, ISO 27002, ISO 20000
President, Slater Technologies, Inc.*

Wireshark/LUA

This article explores an extension mechanisms offered by Wireshark. After a brief description of Wireshark itself, it shows how Wireshark can be extended using Lua as an embedded language. It shows the benefits to be gained from using the combination of Wireshark and Lua. Next, the article explores a way to extend Lua with C code. It shows how Lua can be leveraged by using functions implemented in plain C.

Caveat: The focus of this article is the Wireshark/Lua interplay and the Lua/C interplay. Descriptions of Wireshark as a network analyzer, or Lua and C as programming languages are out of scope for this article.

Wireshark

Wireshark is the de facto industry standard for network protocol analysis. To say it with the words of wireshark itself: “Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

(http://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs retrieved on Oct, 11th 2012)” The open source product successfully overtook commercial competitors. The wireshark’s playground is network communication in all its glory. Protocol analysis typically consists of two separate steps: harvest and analysis. Prior to analysis we need to harvest things to analyse. Wireshark outsources this task to external libraries (WinPcap for Windows, libpcap for other OS). These libraries implement the pcap API. Wireshark grabs network communication using these libraries and writes it to disk. Once network communication has been harvested we end up with files containing raw binary data (also known as traces or dumps). This data contains all the secrets we might ever want to know. Unfortunately, the format is somewhat *unwieldily*, hard to understand

and as efficient for network communication as unsuitable for human consumption. This is where Wireshark displays his real strength: It splits any given dump into single packets (also known as frames), dissects the different protocol layers of any given frame, and displays the protocol tree and all the fields contained within the different protocols in a human readable user friendly format.

Benefits

Wireshark successfully bridges the gap between a machine friendly efficient binary representation of network communication and mere mortals. To illustrate this point in brutal clarity, we compare the raw view on the data with the wireshark view. As an example we take a http GET requests to [http:// http://hakin9.org/](http://http://hakin9.org/): Figure 1.

```

0000 00 03 c9 b0 9c 28 64 31 50 90 79 ac 08 00 45 00 .....(dl P.y...E.
0010 02 62 02 8a 40 00 80 06 75 6c c0 a8 02 82 4f 7d .b.,@., u.....D)
0020 6d 18 1d 3c 09 10 6c 8a 2c 06 0f 13 84 89 50 85 ..,.,P. 5.....P
0030 80 00 75 2a 00 00 47 45 54 20 2f 20 48 54 54 50 ..u.,GE T / HTTP
0040 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 68 01 0b 69 ./A.,.no ect:hafI
0050 6e 39 2e 6f 72 67 0d 0a 43 6f 6e 6e 65 63 74 69 n9.org., Connect
0060 0f 6e 3a 20 6b 65 65 70 2d 6c 6c 69 76 65 0d 0a ont: keep-alive..
0070 41 63 63 65 70 74 3a 20 2a 2f 2a 0d 0a 55 73 65 Accept: */*.Use
0080 7d 41 67 65 6e 74 3a 20 0d 6f 7a 69 6c 61 f-agent: Mozilla
0090 2f 35 2e 30 20 28 57 69 6e 64 6f 77 73 20 4e 54 /s.0 (wi ndows NT
00a0 20 35 2e 31 29 20 41 70 70 6c 65 67 65 62 4b 69 5.1) Ap plewebkfi
00b0 74 2f 35 33 37 2e 34 20 28 49 88 54 6d 6c 2c 20 t/537.4 (ntnM.,
00c0 6c 69 6b 65 20 47 65 63 6b 6f 29 20 43 68 72 6f 13ke gec ko) Chro
00d0 6d 65 2f 32 2e 30 2e 33 32 32 39 2e 39 34 20 me/2.0.1229.9e
00e0 53 61 66 61 72 69 2f 35 33 37 2e 34 0d 0a 41 63 Safari/5 37.4..Ac
00f0 63 65 70 74 2d 45 6e 63 6f 64 69 6e 67 3a 20 67 cept-enc oding: g
0100 7a 69 70 2c 64 65 66 6c 61 74 63 2c 73 64 63 68 2ip,defl ate,sdch
0110 0d 0a 41 63 63 65 70 74 2d 6c 61 6e 67 75 61 67 ..Accept -Languag
0120 65 3a 20 64 65 2d 44 45 2c 64 63 30 71 3d 30 2e 67: 8e-0e..de-jp-0.
0130 38 2c 65 6e 2d 55 53 3b 71 3d 30 2e 36 2c 65 6e 8,en-us; q=0.6,en
0140 3b 71 3d 30 2e 34 0d 0a 41 63 65 70 74 2d 43 :q=0.4.. Accept=
0150 68 61 72 73 65 74 3a 20 49 53 4f 2d 38 38 35 39 harsset: ISO-8859
0160 2d 2c 75 74 6e 6b 38 3b 71 3d 30 2e 38 37 2c 2a -.1,utf-8 1=q0.7,*
0170 3b 71 3d 30 2e 33 0d 0a 43 6f 6f 69 65 3a 20 :q=0.3.. cookie:
0180 41 50 6e 69 73 69 74 6f 72 49 64 5d 37 64 65 PAdnI51t.0rt=ide
0190 02 30 36 64 66 37 61 30 61 66 39 39 63 63 60 65 b06f7a0 8f0e=C0e
01a0 33 33 38 66 36 34 37 32 33 31 30 6f 44 36 20 5f 338f6472 3100e;
01b0 0f 75 6d 61 3d 39 38 33 36 32 34 36 2e 31 ..utm=98 362246..T
01c0 30 33 35 31 31 36 36 32 35 2e 31 33 34 39 39 34 03511662 5.134994
01d0 55 37 38 2e 31 33 34 39 34 35 37 38 34 2e 31 3b 20 5f 5f 5704.134.9945784.
01e0 31 33 34 39 39 34 35 37 38 34 2e 31 3b 20 5f 5f 13499457 84.1.;
01f0 75 74 6d 62 34 39 38 33 36 32 34 36 2e 31 3b 20 5f 5f utm=985 62246..T.
0200 31 30 2e 31 33 34 39 39 34 35 37 38 34 3b 20 5f 5f 10.13499 45784.;
0210 5f 75 74 6d 63 3d 39 38 33 36 32 32 34 36 3b 20 ..utm=98 362246;
0220 5f 75 74 6d 7a 3d 39 38 33 36 32 32 34 36 2e ..utm=9 8362246.
0230 31 33 34 39 39 34 35 37 38 34 2e 31 2e 31 2e 75 13499457 84.1.;u
0240 7a 6d 63 73 3d 28 64 69 72 65 63 74 39 7c 75 tmcnm=d frectj|u
0250 74 6d 63 6e 3d 28 64 69 72 65 63 74 29 7c 75 tmcnm=d frectj|u
0260 74 6d 63 6d 64 3d 28 6e 6f 6e 65 29 0d 0a 0d 0a tmcmd=(one)....

```

Figure 1. raw view

The expert might notice the beginning of the IP header (hex: 45 00) in postion 14. Reading hex, however, soon becomes inefficient and boring. Thus, a more human-friendly representation of the information contained in the raw data is what we really need. This is exactly where Wireshark helps (Figure 2).

```

# Frame 2 (624 bytes on wire (484 bytes captured)
# Ethernet II, Src: 0a:13:15:00:79:ac (0a:13:15:00:79:ac), Dst: TechnoBoh128 (00:03:c9:00:9c:12:8)
# Internet Protocol, Src: 192.168.0.210 (192.168.0.210), Dst: 70.32.106.24 (70.32.106.24)
# Transmission Control Protocol, Src Port: 7316 (7316), Dst Port: Http (80), Seq: 1, Acl: 1, Len: 170
# Hypertext Transfer Protocol

# GET /HTTP/2.0/ HTTP/2.0
  Request Method: GET
    Request URI: /
    Request Version: HTTP/1.1
    Host: hakin9.org/v/n
    Connection: keep-alive/v/n
    Accept: */*v/n
    user-agent: Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.4 (KHTML, like Gecko) Chrome/20.0.1229.94 Safari/537.4
    Accept-encoding: gzip, deflate, sdch/v/n
    Accept-language: en-us;q=0.6,en;q=0.4,vn;q=0.4
    Accept-charset: ISO-8859-1,utf-8;q=0.7,*q=0.3v/n
    [truncated] cookie: hakin9.org=5e0c6fPaa9Pcc0e338F6f732300; ..utm=98362246.10.1349945784.1.;u
    v/n

```

Figure 2. Dissected view

The raw binary data is analyzed and the onion like structure of the protocol tree is unwrapped and displayed in an expandable tree like fashion. This way wireshark enables the human reader to have a clear view on the protocols and fields of each and every packet contained in a given trace. Apart from this core functionality, Wireshark overwhelms the user with a

plethora of advanced analysis features. These features are out of scope for this article. Now that we can easily see the complete communication contained in a given trace we can easily answer each and every question that might come into our mind – at least if we know the intricacies of all protocols involved in the trace.

Limitations

Wireshark is the tool of choice for manual expert analysis of trace files. This core capability also directly leads us to two major areas of concern: the analysis is manual and has to be done by experts.

Wireshark is not ideally suited for automation, but is mainly conceived for interactive use. As an example, guiding us through the rest of this article, we look at a simple question that is as typical as harmless. Let's assume we have a trace containing plenty of TCP/IP traffic and we are interested in the duration of connection establishment (“RTT from 3WHS, Roundtrip time from three way handshake in tcptrace (see <http://www.tcptrace.org/>, retrieved Oct 11th 2012) lingo”).

The answer of course is simple. We briefly look into the relevant RFCs and soon find out that all we have to do is to calculate the timespan between the first syn request and the ack request from the counterparty. We can accomplish this interactively by using the “Follow TCP Stream” feature of Wireshark and doing our little math. We set the time display format to “Seconds since Beginning of Capture” and subtract the time value of the syn requests from the value of the ack request. This is fine for a single TCP session or a smallish number of sessions. It soon becomes tedious once the number of sessions rises.

Of course, there is an obvious improvement to this approach. We soon befriend Wireshark's batch cousin tshark, do some fancy filtering, pipe the result into a shell script and do our math in the shell script. As this becomes hard to maintain, we substitute the shell script with a script language of our choice. Now we already need Wireshark, a suitable interpreter and our script to do our analysis. Alternatively, we could resort to tools like tcptrace and parse and process the results.

From an engineering point of view, these solutions are workable and pragmatic but less than elegant. The engineer would prefer an integrated solution to this exemplary problem.

Lua

This is where Lua (Portuguese for “Moon”) enters the fray. Lua is a small and fast script language that is embedded into Wireshark. We can use it to automate Wireshark. In order to use Lua from within Wireshark, we first check if our particular Wireshark instance has been compiled with Lua support (Figure 3).

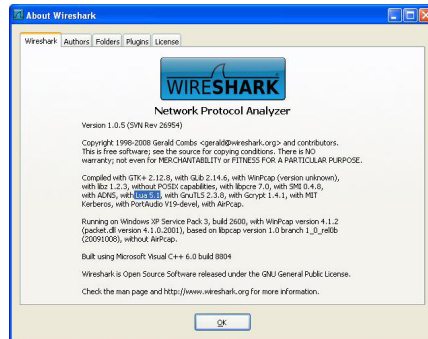


Figure 3. Help-> About Wireshark

In the About Dialog we verify that our particular Wireshark has been compiled with Lua support. We are now ready to go.

The language

Let us introduce Lua in its own words: “Lua is an extension programming language designed to support general procedural programming with data description facilities. (...) Lua is intended to be used as a powerful, light-weight scripting language for any program that needs one.”

(<http://www.lua.org/manual/5.1/manual.html>, retrieved Oct 11th, 2012).

The Lua interpreter is contained within Wireshark. This means we do not need any external interpreter or other external tools. Any solution build upon Wireshark and Lua runs stand-alone without external dependencies. This considerably improves the robustness of any such solution and considerably eases deployment.

Overcome Wireshark limitations

We now have the means to overcome Wireshark’s limitations. We can codify expert know-how using the Lua language. Within the embedded Lua language we have full access (well, nearly full) to Wireshark capabilities. We can now accomplish typical batch processing tasks without resorting to shell scripts or external script languages. Using Lua we have the benefit of a clean API to access Wireshark capabilities instead of piping the results of a

Wireshark processing step into an external process. The beauty of this approach consists of the chance of combining the strength of frame/packet oriented dissectors with the capabilities of a full programming language without incurring the extra cost of additional dependencies.

Real world example

The example from above (RTT from 3 WHS) may serve as our real world example. It shows the mechanics of Lua programs running embedded within Wireshark.

First, we identify a script named “init.lua” and follow the advice given in the header section: “Lua is disabled by default, comment out the following line to enable Lua support.” We bravely comment out the line reading `disable_lua = true; do return end;` and proceed (Figure 4).

```
1 local tap_tcp = Listener.new({})
2
3 local synflag = field.new('tcp.flags.syn')
4 local ackflag = field.new('tcp.flags.ack')
5 local src = field.new('tcp.src_ip')
6 local dst = field.new('tcp.dst_ip')
7
8 local map = {}
9
10 -- calculating rtt as delta between syn and ack
11 function tap_tcp.packet(pkt, info)
12     local src = string.format("%s:%s", info.src_ip, info.src_port)
13     local dst = string.format("%s:%s", info.dst_ip, info.dst_port)
14     local reverse = string.reverse(src)
15     local reverse_dst = string.reverse(dst)
16     local rtt = info.time_delta
17     local time = string.format("%s", os.date("%b %p", pinfo.abs_ts), string.sub(pinfo.abs_ts, 12))
18
19     -- Is there a syn?
20     if (synflag() and tostring(synflag())=="1" and ackflag() and tostring(ackflag())=="0"
21         and src() and tostring(src())=="*" then
22         map[src] = (pinfo.number, pinfo.abs_ts)
23     end
24
25     -- Find ack, calculate delta and print out
26     if (ackflag() and tostring(ackflag())=="1" and src() and tostring(src())=="*"
27         and map[src] then
28         local reverse_src = string.reverse(src)
29         local reverse_dst = string.reverse(dst)
30         local rtt = info.time_delta
31         local time = string.format("%s", os.date("%b %p", pinfo.abs_ts), string.sub(pinfo.abs_ts, 12))
32         map[reverse_src] = map[reverse_dst]
33     end
34 end
```

Figure 4. Content of rtt.lua

In line 1 we register a listener for tcp. The callback function `tap_tcp.packet` is invoked for each tcp packet. We can easily access various fields of the packet using the `pinfo` structure. In line 3-6 we directly access Wireshark fields. Wireshark exposes all fields of all protocols using this API. The idiom behind the listener/callback construction is similar to the mechanics of pattern matching tools like `awk`. `Awk` scans text files, checks if a specified pattern occurs within a scanned text file and executes actions registered with certain patterns. The basic mechanism of Lua scripts within Wireshark consists of registered and callback functions that are called whenever a particular listener “fires” while scanning a trace file. We invoke the script with the command line “`tshark -q -X lua_script:rtt.lua -r yourtracefile.pcap`”. The script writes out the frame number of the ack request, source and destination ip, frame number of the syn request, duration of connection establishment and the absolute time of the ack request.

Benefit of team Wireshark/Lua

Using Lua as an extension language embedded in Wireshark gives a number of benefits. To name but a few:

- Tight integration into Wireshark allows access of tons of Wireshark functionality without any further hassle.
- Lua as a full blown language allows any procedural processing we feel obliged to do. This way it is possible to use Wireshark asynchronously in a batch environment.
- Being able to script analyses formerly done in an interactive way allows us to perform the analyses in a more efficient way.
- Putting expert know how in scripts allows non experts to perform analyses.
- The approach works in restricted environments where other languages might not be available

The possibilities shown so far only scratch the surface of Lua/Wireshark integration. Lua can be used to write full blown custom dissectors. The user interface is not limited to the command line. Lua can also be used to access GUI capabilities. Output from functionality implemented with Lua can be rendered by GUI components.

Outlook: extend Wireshark/Lua with C

There are situations where we might feel the urge to access functionality buried in C from within Lua. Either there is existing functionality to be reused or there are challenges more easily solved in C than in Lua.

Warning

Setting up a suitable c compilation environment can pose challenges. A detailed description is out of scope for this article (see http://www.troubleshooters.com/codecorn/lu/lua_c_calls_lua.htm retrieved Oct 11th, 2012 for details). Your mileage may vary. The compilation described below has been tested in a MingW Environment.

After these words of warning we proceed with our endeavor of exposing C functionality to the winning combination of Lua/Wireshark. In order for the compile to succeed it is necessary to put lua header files and lua libraries in directories where the compiler can find them. In case these files live in other directories the compiler has to be informed by suitable compiler switches (-I and -L in case of gcc) of the directories these files live in. It is

all important that header and libraries match with the Lua version used by Wireshark. For Lua 5.1 in Wireshark use Lua 5.1 header and libraries. The header files (lua.h, luaconf.h, lauxlib.h, lualib.h) may live in MingW/include. The libraries (liblua.a, liblua.dll.a) may live in MingW/lib (Figure 5).

```

1  /*
2  * We need to include the standard lua headers
3  */
4  #include <lua.h>
5  #include <lua11b.h>
6  #include <lua11b.h>
7
8  #include <stdlib.h>
9  #include <time.h>
10 /*
11 * our implementation
12 */
13 -static int random(lua_State *L){ /* name of our implementation */
14 /* initialize random generator */
15 srand ( time(NULL) );
16 lua_pushnumber(L, rand());
17 return 1; /* we return one value */
18 }
19
20 /*
21 * register our module, works with naming convention: "luaopen_nameofyourmodule"
22 */
23 -int luaopen_random(lua_State *L){
24 lua_register(
25     L, /* the lua state */
26     "random", /* arbitrary name to use within lua */
27     random /* the name of our implementation */
28 );
29 return 0;
30 }
31

```

Figure 5. *callfromlua.c. Function to be called from Lua*

The custom function to be used from Lua is straight forward. It simply returns a random number. The function has to be registered in the call to `luaopen_*`. This function actually registers each function that is exposed to lua. From within Lua we can access the functionality using the name “random”. We compile the code to a dll using a command like `gcc -Wall -shared -o random.dll callfromlua.c`. This call may vary for your system depending on compiler and environment. The compilation should proceed without any warnings or errors. The resulting dll has to be placed in the Wireshark root directory. We are now ready to play with our C extension (Figure 6).

```

1  require("random")
2  print(random())
3

```

Figure 6. *c.lua. Calling our C function*

First, we require the module implemented in C (line 1). Wireshark looks at several locations for a shared library named like the module – `random.dll` in case of windows. It then loads the library and executed the `luaopen_mondulename` function named like the module and reports an error in case this function is not found. The functions registered by this function – in this case a single function “random” are now available for ordinary Lua code. We simply invoke the custom function implemented in C (line 2). From the Lua point of view using functions implemented in C is similar to

other function calls. A command line like “tshark -X lua_script:c.lua” now prints out our random number generated by C code.

This bare bones example merely illustrates the general mechanics of using C code with Lua/Wireshark. For the sake of simplicity it has been reduced to the essentials.

Where to go from here

We started our exploration with Wireshark as a standard tool for manual expert analysis of network packets. We then explored ways to extend the core Wireshark functionality using the embedded Lua language. Finally, we saw how Lua itself can be extended using C. Using these building blocks we can now go on and leverage Wireshark and automatically perform arbitrary trace analyses using the dissector functionality provided by Wireshark. We can accomplish this without additional external dependencies purely by using functionality offered by Wireshark itself. We can fully automate Wireshark and can use all the functionality in a batch like fashion.

Jörg Kalsbach

Tracing ContikiOs Based IoT communications over Cooja simulations with Wireshark Using Wireshark with Cooja simulator

Internet of Things is getting real. Billions of devices interconnected between each other retrieving data and sharing information using wireless communication protocols everywhere. We present an introduction about how to start developing radio communication applications for Contiki OS, one of the most widespread IoT operating systems and how to use Cooja simulator together with Wireshark.

The number of devices with wireless connection capability has increased over the last years. Nowadays, most of the people deal with the so-called

smart devices, for example, smartphones. However, not only smartphones are able to be connected to Internet, but also a big number of hand held devices such as tablet PC.

Another important trend is related to *Wireless Sensor Network* (WSN), spatially-distributed autonomous devices equipped with several kinds of sensors and interconnected to each other using wireless communication systems. These devices are small-size computers with reduced computation capabilities, which are responsible to retrieve information about its environment and send it to data sinks computers. It is common to refer to WSN as smart dust because of the size of its devices, which are called sensor motes. All those devices are part of the *Internet of Things* (IoT), a scenario where everything is interconnected and identified via Internet, using technologies like IPv6, RFID tags or other systems like barcodes. With the appearance of this concept, we will also be able to communicate with daily use devices, such as the lighting or the heating system available in our house.

Several research works have been performed in order to study the possibilities of this new generation of devices. In fact, related fields such as security, constrained devices properties or communication skills are some of the hottest topics within the researching community.

Regarding to this communication skills, Wireshark has been used as a world-wide network sniffer tool recognising the information exchanged between the elements involved in a network communication. Its use provides us with a clearer way to understand the information exchanged. On the other hand, the motes are small devices that do not include graphical interface in order to facilitate the interaction user-mote. Thus, becoming developers of embedded applications, in other words, applications specifically designed for IoT devices, we need a way to check their correct functioning. A simulator is used to mimic the working mode of a embedded application within a constrained device. However, when the application simulated involves network communication between different nodes, the use of Wireshark in conjunction with the simulator allows a more understandable way to check the correcting communications conducted.

Given that, in this article we present deeply the Internet of Things concept. The deployment of a constrained *Contiki OS* based application within a *Cooja* simulated IoT device is one of the main points in this work. Thus, a

brief overview of *Contiki OS* and *Cooja* is pointed out. Finally, a communication embedded application is set using the simulator and allowing us to get the messages exchanged in different formats. The messages exchanged data is handled by some methods explained in this article, getting in this way different *Wireshark* visualizations. Finally, the article finishes with a set of conclusions regarding to the whole work carried out.

CONTIKI OS

IoT devices are resource constrained devices. In fact, within their features it is worthy highlighting the constraints in the communication skills available as well as computation performance. In addition, the memory available either ROM or RAM, is considerably smaller than the memory sizes we are used to deal with in general purpose computers.

Given those features, there are several dedicated operating systems that help the programmers to face up the challenges found on constrained devices. In the deployment outlined in this article, we will work with *Contiki OS*, an open source operating system for the Internet of Things. *Contiki OS* allows tiny, battery-operated low-power systems to communicate with Internet. Within *Contiki OS*, several platforms are available. Although some of those platforms are embedded platforms such as Micaz, Redbee-Econotag or Sky, there are also available platforms that can be simulated in a PC: minimal-net and Cooja. Thus, if we develop an embedded application and there is no possibility to use a physical device to test the software, a PC-based simulation can be performed. In fact, this is the case outlined in this work, where the simulations of already deployed embedded applications will be performed within *Cooja*, a PC-based simulator for the Internet of Things. Regarding to each platform itself, *Contiki OS* provides us with a framework to work with the different hardware elements available in them. Thus, using this framework we can handle the resources available such as leds and wireless radio. In fact, within this work we will focus in this wireless radio connection, with which we will perform different examples in several uses cases. Besides, the information exchanged between the different simulated nodes can be traced by using the well-known sniffing traffic network tool *Wireshark*. However, before that it is worthy knowing a bit more about how the communication is performed between these constrained devices.

Communication protocol stacks

The communication of embedded devices is performed in a different way to how traditional communication is performed. As its own name indicates, the Internet of Things devices are communicating each other based on IP. However the underlayer configuration is different in order to fulfil the requirements given by the scarce resources available.

Thus, the physical layer as well as the link layer are deployed following the 802.15.4 definition instead of Ethernet, WiFi or WiMax. This new layer configuration will result in a different format in the message exchanged during the communication between the devices. On the other hand, the rest of the stack remain the same.

Within the *Contiki OS*, this new communication protocol stack has been developed by the called microIP stack (Figure 1).

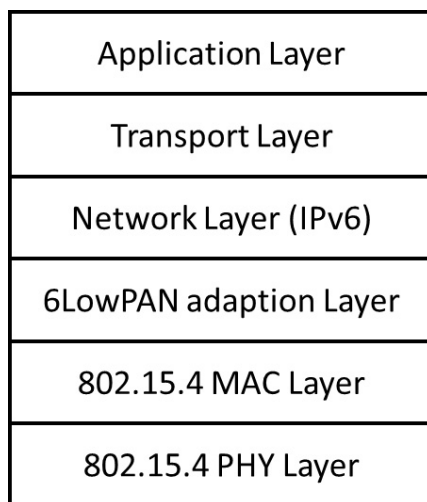


Figure 1. Representation of the microIP stack

In this stack, apart from the above explained modification based on 802.15.4, the 6LoWPAN adaptation layer has been added. This new layer is used for adapting the whole IP layer to a suitable lightweigh-version within the constrained environments. Thus, the main feature of this a IP adaptation layer is to compress the IP headers in order to make the whole packages as small as possible to be sent over 802.15.4 based communications.

This feature is essential in order to understand the whole format of a packet exchanged in this new type of constrained networks. This packet format will lead most part of the work described in this article. Thus, it becomes important to make clear this format itself.

Cooja

Cooja is a simulator of sensor networks for *Contiki OS*. This java based application allow us to simulate embedded applications over different platforms such as Cooja, Sky or Micaz. The main parts of this simulator are the interfaces and the plugins.

On one hand, *Cooja* interfaces involves several graphical representations, where information and interaction with the user is offered. Thus, most of the simulated elements available in a constrained devices can be handled through these interfaces: leds, radio communication module or serial port communication are some examples of interfaces available. On the other hand, *Cooja* plugins are the best way for a user to interact with a simulation. These plugins, implemented as regular Java Panel, allow the user to control the whole simulation itself. One of this *Cooja* plugins is the called Radio messages. This plugin will allow us to extract the information exchanged in a simulated embedded communication and work with it in order to get a representation with *Wireshark*, as we will see later on this document.

First steps in Cooja

How to start

Before installing it, Java 1.6 or later is required on the system. *Cooja* is included in *Contiki* source tree since version 2.0. We can find this simulator in `[Contiki Folder]/tools/cooja`. Once we are within this folder, we have to compile and execute it throught an Ant script:

```
$ ant run
```

Once it is open, we want to execute a hello world example. Go to File menu/New simulation/Create. As a result, a new simulation without any mote and using default parameters will appear. We want to run a simulation in a specific type of mote, then we need to create that mote and load the program on it. We use *Cooja* type mote here because all the programs should run on it: Motes menu/Add motes.../Create new mote type/Cooja mote...

Then we have to choose the program we want to execute: click on Browse and go to `[Contiki folder]/examples/hello world/hello-world.c`, then press Compile. This process will compile the whole Contiki OS and the application, creating just a file `hello-world.cooja` that contains both the OS and the application. Last step requires us to introduce the number of motes for the

simulation, then click on Add motes. In this case just one mote is enough. Once the simulation is ready, just click on Start and we will see the output in the Mote output window (Figure 2).

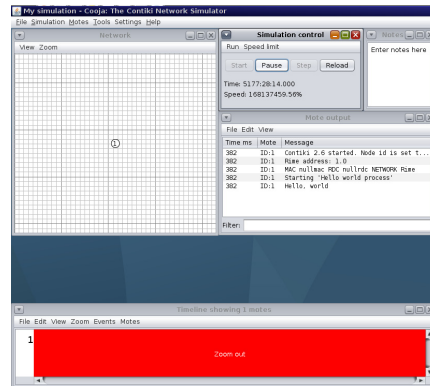


Figure 2. Hello world example simulated in Cooja

The environment

When creating a new simulation, several properties can be modified. It is possible to modify the radio medium, the motes startup time and also the random seed for the random number generator. By default, there are some kinds of motes available, included Sky mote, Micaz and also a general one called Cooja mote, but it is also possible to extend Cooja simulator in order to introduce different platforms. Simulations can be exported, saved and loaded. Simulations can be automatized using shell scripts that also retrieve the data after perform the simulation. *Cooja* includes a toolbox that aid to perform the simulations and gather data from them:

- simulation control tool allows to set simulation speed,
- mote output shows all the data from the serial port,
- event listener helps establishing break points in the simulation,
- radio messages captures radio communication between motes and allows to export those captures,
- mote radio duty cycle allow performing measurements about the radio utilization on a device,
- the simulation visualizer window shows the simulation behaviour and allows to show different information about the motes being used such as LEDs or radio information,
- finally there is a timeline component which shows the different events in the simulation among the existing motes.

In summary, *Cooja* is a very useful tool in the design phase of *Contiki OS* applications. It can deal with different kind of platforms and it is extensible. Thus, it is a very useful tool to deploy embedded applications and check them within simulated constrained devices.

How to set a Communication Simulation

Client – server

The first communication based basic program available as an example in Contiki involves a client and a server exchanging information over UDP. This example shows us how a UDP based communication is performed by using microIP stack. Thus, it becomes in a good example to see how Wireshark traces are obtained within this environment and how they can be managed.

How to write the code

Taking a look of the code of both client and server, a similar structure is defined. The most important functions are:

- `tcpip_handler()`. This is used for handling the messages received through wireless radio communication. At this point, two main variables are taken into account: `uip_appdata`, a pointer to the buffer with the received information and `uip_datalen()`, a function returning the length of the message received.
- timer related functions. A timer is used in the client to send a message to the server every time the timer is expired. Thus, it is essential to handle also several timer related functions such as `etimer_set()`, `etimer_expired()` and `etimer_restart()`.
- `timeout_handler()`. Once a timer is defined, a corresponding handler has to be defined as well. In the example that we are using, the related handler is the `timeout_handler()` function. In this function, a message is created and sent to the other communication end.
- `set_connection_address()`. This essential function is used for setting up the IP address of the other end in the communication. Thus, in the client's code, the server's IP address has to be correctly set and viceversa.
- `uip_udp_packet_send()`. A function called to send a message over the wireless connection established. If every parameter is previously correctly configured, the message included in this function call will be sent to the other end within the communication.

With these essential and simple functions, a main client and server programs can be developed. The complete C code of those programs can be found in [Contiki Folder]/examples/udp-ipv6.

How to Simulate

Previously in this article, a simulation of the helloWorld embedded application has been outlined. In order to create a simulation containing the UDP client and the UDP server, the same basic steps have to be followed for each application.

Thus, a new simulation has to be created. Within this simulation, two new Contiki type motes should be added. In one of them, the udp-client.c application is loaded whereas in the other mote the udp-server.c must be loaded. If every step has been successfully performed, a simulation containing both elements, client and server, should be correctly showed (Figure 3).

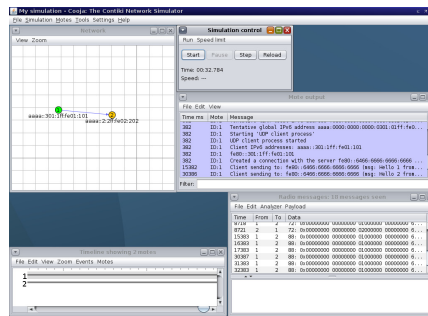


Figure 3. Client-server scenario simulated in Cooja

At this point, if the simulation is executed, the client will keep on sending messages to the server, but they will not reach it. This will happen because the IP address set in the [Contiki Folder]/examples/udp-ipv6/udp-client.c, within the `set_connection_address()` function, is not correct. In order to fix it, we should check the IP address of the server in our Cooja simulation and set it in the `udp-client.c` program. Once we have the server's address just go to `set_connection_address()` function and modify `uip_ip6addr()` function's parameters. In our case, the IP address assigned to the server is `aaaa:301:1ff:fe01:101`, so the function invocation is `uip_ip6addr(ipaddr, 0xfe00, 0, 0, 0, 0x301, 0x1ff, 0xfe01, 0x101)` (Figure 4).

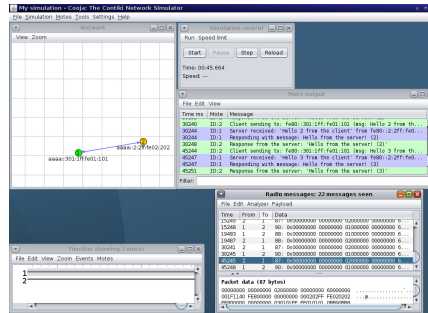


Figure 4. Client-server fixed scenario simulated in Cooja

How to log the messages

Once the simulation is working properly, we have the opportunity of extracting the Wireshark traces of the communication performed between the client and the server. For this purpose, the first step is to reload the simulation to get it as a new one. Thus, click on File/Reload simulation/new random seed. The whole simulation will be loaded again.

Once the simulation is correctly loaded and before starting the simulation, we need to set up the plugin to capture the messages exchanged in the communication. For this purpose, we should click on Tools/Radio messages. A new window will appear. In this Radio messages window, a representation of the messages exchanged in the communication will be stored.

Now we can start the simulation and we will see that the client and the server are correctly sending messages each other through two interfaces available. On one hand, in the Mote output window, the log of both applications will appear. On the other hand, in the Radio messages window, the hexadecimal representation of the messages will be logged as well. After some simulation time, when some messages are exchanged between the client and the server, the simulation can be stopped. Now, we are ready to export our simulated communication to a Wireshark format.

How to see the messages in Wireshark

The Radio messages plugin allow us to export the hexadecimal based communication log to a pcap format, which is recognized by *Wireshark*. In order to get that, once the log has been collected in the Radio messages plugin, we should click on Analyzer menu and select 6LoWPAN Analyzer with PCAP. In this moment, a *Wireshark* trace is created with every message exchanged between the two motes.

This new trace can be found under *[Contiki Folder]/tools/cooja/build/*. It will be called radiolog-xxxxxxx.pcap, where the x are substituted by numbers. This file can be directly opened using *Wireshark* application. We will obtain a trace as depicted in fig. In this trace we can see how every message is defined as 802.15.4 message (Figure 5).

A 802.15.4 based network behaves like a general purpose network. Thus, before the messages containing the data Hello from the client and Hello from the server appear in the communication, other set of 802.15.4 messages are exchanged in order to establish the network communication itself. We can compare this previous messages exchanges with the ARP mechanism deployed in general purpose networks in order to discover the addressing information related to the network peers.

Once the 802.15.4 network is established, we will be able to see client and server application data within the messages depicted in *Wireshark* trace.

How to format messages following the traditional IP stack

The output obtained directly from the Radio messages plugin is not easily understandable. Opening the trace obtained with *Wireshark* application, we can observe different messages composed by an 802.15.4 header carrying some data. However, it can be formatted in order to get a more understandable format of the application data exchanged.

For this purpose, the first step to perform is to obtain the raw data exchanged instead formatted as pcap. This can be done by selecting File/Save to file option in the Radio messages. We save the raw data application exchanged in a file, in this case called output. If we open this output file, a hexadecimal representation of the 802.15.4 messages is depicted. However, we want to have them following the traditional IP stack. Thus, the next step is to format every message in order to get only the UDP and application parts of the message. In order to get this, we need to take into account in which byte position the UDP related information starts within the message.

Knowing that, we will format the messages previously saved in the output file in order to keep just their UDP and application related data. Besides, a set of zeros need to be set at the beginning of the message in order to simulate its sequence number as expected by *Wireshark* application. The step described above can be done using this C++ code (Listing 1).

Listing 1. Parser from Cooja to Wireshark

```

#include <iostream>
#include <string>
#include <cstring>
#include <stdio.h>
using namespace std;
#define POS_INIT_UDP 113
int main (){
string str;
while (getline(cin, str)){
cout << "000000 ";
for(int i=2; i<str.size();i++){
if (i>POS_INIT_UDP) {
cout << str[i];
if(i%2)
cout << " ";
}
}
cout << endl;
}
}
}

```

Assuming that we save this code in a file called `parser-from-cooja.cpp`, we compile this C++ code by using the next command line:

```
g++ parser-from-cooja.cpp -o parser.out
```

In this point, we have the parser needed for extracting a file with every message parsed. Thus, if we apply directly this parser to the output file we will obtain messages tailed with the UDP and application data only. To get this tailed file we can perform

```
sudo chmod 777 ./parser.out; ./parser.out < output
```

However, this remains to be in a incorrect format understandable by *Wireshark* application. Thus, we need to add the underlayer headers to these messages in order to get them over a simulated traditional communication stack. In other words, we need to simulate that the message has been exchanged by using the following underlayer headers: ethernet, IP, UDP, application data.

For this purpose we can use the next bash script:

```

cut -f2- -d " " < output | tr -d " " |
./parser.out > delete_wireshark_temp && text2pcap
-o hex -i 17 delete_wireshark_temp out && wireshark out

```

This script parses the raw output obtained from the Cooja plugin called Radio messages, obtaining the file `delete_wireshark_temp`. Within this file we have a representation of every message containing just their UDP and application layers. After that, with the GNU/Linux tool `text2pcap`, we will simulate a IPv4 stack. By indicating that the Next Header is a UDP header (option `-i 17`), this tool will create this simulated IPv4 stack and it will

append the UDP and application data contained within the `delete_wireshark_temp` file.

Finally, the *Wireshark* application will be opened and then every messages is depicted as an UDP message. As explained before, several messages are exchanged in order to set the network in which our simulated nodes are exchanging information. In order to check the messages in which we are interested, we should look for those which UDP port numbers are 3000 and 3001. Those messages are the ones exchanged between *udp-client* and *udp-server*. Actually, as depicted in Figure 6, we can see how the string Hello from the client can be correctly be watched in the *Wireshark* application.

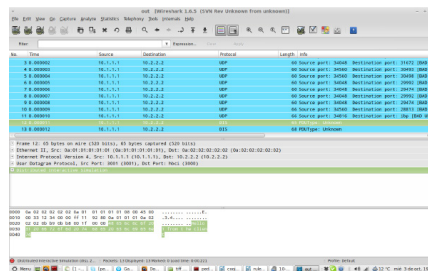


Figure 6. *Wireshark* trace showing UDP/IP based messages

Conclusions

In this work we present an overview of the recently appeared work of Internet of Things. Developing embedded applications for embedded devices is a task that can be helped by using a simulator. *Cooja*, the simulator described within this work, allow the developer of constrained applications to check their correct functioning given the lack of graphical interfaz in IoT devices. The *Cooja* environment presented in this article will allow the reader to simulate his first embedded application as tutorized within this work. Finally, a deep handling of the *Wireshark* application in conjunction with the simulations carried out, show how this world wide known application is applicable in this new area. In addition, handling the associated message information allows the developers to get a more understable and totally configurable output within the *Wireshark* application. Thus, the IoT background, the simulation procedures as well as the *Wireshark* related techniques presented in this work aim at becoming in a referencing start point for those developers who want to create their own constrained applications.

On the Web

• <http://www.contiki-os.org/> – Contiki operating system main page

- http://wiki.contiki-os.org/doku.php?id=an_introduction_to_cooja – Introduction to Cooja simulator
- <http://www.wireshark.org> – Wireshark official web page

Pedro Moreno-Sanchez

Pedro Moreno-Sanchez. M.Sc. student at the University of Murcia, Spain. His background is related to IP-based security protocols. Nowadays, he is directly involved in the project OpenPANA: An opensource implementation for network access control based on PANA.

Rogelio Martinez-Perez

Rogelio Martinez-Perez is a BCS in Computer Science at the University of Murcia, Spain. He has experience in working on the Internet of Things and Smart Sensor Networks.

Lint Center

for National Security Studies, Inc.™

EMPOWER, ENHANCE, ENABLE

Need a scholarship?

White hats, Ninjas, Grinders, and Engineers – listen up!

The Lint Center for National Security Studies awards merit-based scholarships semi-annually in both July and January. A streamlined, web-based application form is available on our main portal. Undergraduate and post-graduate students pursuing technical degrees in computer security, computer science, diplomacy, and linguistics are encouraged.

LintCenter.org

About the Lint Center: The Lint Center for National Security Studies in the United States is a Veteran and Minority directed, all-volunteer 501(c)(3) non-profit organization, dedicated to fostering the educational development of the next generation of the National Security and Intelligence communities by providing passionate individuals with scholarship opportunities and mentorship from experienced National Security personnel.

About the Lint Center's Mentoring Program:

In addition to the scholarship award, winners will acquire an experienced security practitioner-mentor. With over 150 mentors, the Lint Center is well positioned to match emerging leaders with practitioners to streamline the learning curve.

Check out our blog: LintCenter.info

Follow us on Twitter: [@LintCenter](https://twitter.com/LintCenter)

Become a fan: facebook.com/LintCenter

EMPOWER, ENHANCE, ENABLE...

(Script Kiddies need not apply)

Learn ethical hacking > **Become a Pentester™**

- ✦ Get trained today through our exclusive 7-months hands-on course.
- ✦ Gain access to our complex LAB environment exploiting vulnerabilities across many platforms.
- ✦ Receive a trainer dedicated to you during the 7 months.
- ✦ 10 different hands-on engagements, 2 different certifications levels.

MONTH 1	<ul style="list-style-type: none">> Vulnerability Assessment - level 1> Vulnerability Assessment - level 2> Vulnerability Assessment - level 3
MONTH 2	<ul style="list-style-type: none">> Network Penetration Testing - level 1> Network Penetration Testing - level 2
MONTH 3	<ul style="list-style-type: none">> Network Penetration Testing - level 3
MONTH 4	<ul style="list-style-type: none">> Web Application Penetration Testing - level 1> Web Application Penetration Testing - level 2
MONTH 5	<ul style="list-style-type: none">> Web Application Penetration Testing - level 3
MONTH 6	<ul style="list-style-type: none">> Certification Exam 1 - Certified Cyber 51 Pentesting Professional - (CC51PP)
MONTH 7	<ul style="list-style-type: none">> Certification Exam 2 - Certified Cyber 51 Pentesting Expert - (CC51PE)

~~Regular Price~~
1260 USD

Discounted Price
999 USD

Sign Up Now

www.cyber51.com





[GEEKED AT BIRTH.]

PWR: 110%

IM Geek PH: 877.UAT.GEEK

[IT'S IN YOUR PULSE.]

LEARN:

Advancing Computer Science
Artificial Life Programming
Digital Media
Digital Video
Enterprise Software Development
Game Art and Animation
Game Design
Game Programming
Human-Computer Interaction
Network Engineering

Network Security
Open Source Technologies
Robotics and Embedded Systems
Serious Game and Simulation
Strategic Technology Development
Technology Forensics
Technology Product Design
Technology Studies
Virtual Modeling and Design
Web and Social Media Technologies



You can talk the talk.
Can you walk the walk?

www.uat.edu > 877.UAT.GEEK

PLEASE SEE WWW.UATEMFASTRACTS FOR THE LATEST INFORMATION ABOUT DEGREE PROGRAM PERFORMANCE, PLACEMENT AND COSTS.