Delft University of Technology

CS4110 Software Testing and Reverse Engineering

Malware Anaysis

z3r0_t0_h3r0.NET

Students:	**Student ID:**
Ginger Geneste	4081315
Marijn Goedegebure	4013484
Sille Kamoen	1534866
Harmjan Treep	4011724

April 10, 2016

Contents

1 Introduction

For the course Software Testing and Reverse Engineering the students have been tasked to analyse a Zeus binary and extract the encryption key. Zeus is a Trojan horse malware package that used to be very popular amongst cyber criminals. It offers a wide range of modules, each used for different malicious purposes such as credential theft. At its peak in 2009, millions of PC's were infected[1]. The malware focused on stealing bank credentials, and proved to be very effective. Damages are estimated to be hundreds of millions of dollars stolen from companies, banks and individuals; more then all other credential-stealing malware combined. Due to the many stealthy techniques used and the constant updates, Zeus was found hard to detect. By generating a unique version of the binary for each "customer" that bought Zeus, traditional detection techniques were not very effective.

In May 2011, the source code of Zeus was leaked online. The exact reason or cause for this remains shrouded in mystery, as it is speculated to be a diversion to hide the release of a new type of malware. The release has both good and bad sides. It caused a surge of new versions and infections, but it also allowed indepth analysis. For different purposes throughout the project, the team has also looked into this source code as provided on GitHub. [2].

Zeus encrypts messages between the infected PC and the Command and Control (C&C) server with the RC4 encryption algorithm. RC4 (Rivest Cypher 4) is a streaming cypher which is known from the unsafe WEP protocol, which is based on RC4. RC4 uses a single key to generate the streaming cypher key. This means that, if you have the original private key, you can decrypt any message sent by the malware. This fact has resulted in successful analysing efforts in the past, such as the research done by Caballero et. al. [1]

Goal

The primary goal of the exercise is to gain experience with malware analysis. As a first step, the aim is to extract the RC4 key from the executable. After the key extraction, the next step is to analyse the server side of Zeus. The server, which handles the command and control, will be set up in a private network. The goal of the analsysis is to gain knowledge on how the back-end of this malware functions and how it is maintained.

2 Background on Malware Analysis

The purpose of malware analysis is to gain a better understanding of how software with a harmful intent functions in order to improve the current defences. Analysing malware can be performed using two different approaches: *static* analysis and *dynamic* analysis. As explained by Egele et al.[2], static analysis involves code analysis without executing the software whereas dynamic analysis investigates the behaviour of the software while it is being executed. While one could perform only one of the two malware analysis approaches, it is highly recommended to take both dynamic as well as static analysis into account in order to get a good understanding of the malware that is being analysed.

During the first phase of this project, the team has investigated nine different studies of interest related to malware analysis and in specific botnets. The studies have been briefly summarised and submitted for the first assignment. These studies gave us insights on interesting approaches for malware analysis and a general understanding on the topic.

Most of the studies the team has investigated, analysed a type of malware that is called a *botnet*. A botnet is a collection of infected clients controlled by command and control (C&C) servers. As depicted in figure 1, a client gets infected upon the retrieval of malware. The malware turns the infected client into a bot which means it will honour the commands sent from a C&C server. These commands could include malware updates, infecting other clients, sending data it has gathered and many others. Once a client has joined a botnet, it is most likely to get infected with the piece of malware the botnet was mainly designed for. For example, Zeus is a well known botnet, and infects its clients with the Zeus banking malware. While this is the main purpose of the botnet, a bot within the Zeus botnet could also serve many other purposes as the botnet owner has complete control of the infected machines.

[1] `http://www.bloomberg.com/news/features/2015-06-18/the-hunt-for-the-financial-industry-s-most-wanted-hacker`
[2] `https://github.com/Visgean/Zeus`

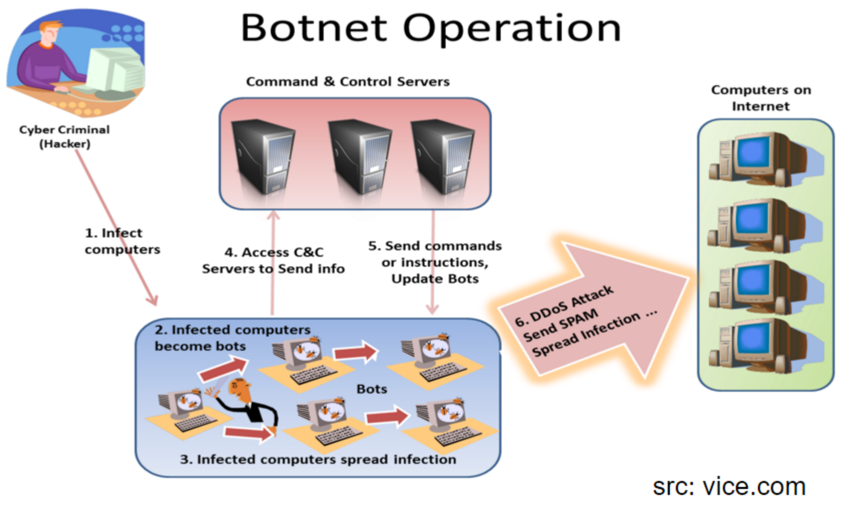

Botnet Operation
Cyber Criminal (Hacker)
Command & Control Servers
Computers on Internet
1. Infect computers
4. Access C&C Servers to Send info
5. Send commands or instructions, Update Bots
6. DDoS Attack Send SPAM Spread Infection ...
2. Infected computers become bots
Bots
3. Infected computers spread infection
src: vice.com

Figure 1: A graphical representation of a botnet

3 Approach

The team has been provided with an executable of the leaked version of Zeus by the university for research purposes. As this is a working sample of actual malware, to analyse the behaviour of Zeus, both caution and precision are required.

Many different approaches exist for analysing malware and many of these have been discussed in the paper by Egele et al.[2]. The work mentions techniques such as function call monitoring, function parameter analysis, information flow tracking, instruction trace and autostart extensibility points and these have been used as a guideline for this project. The paper also gave insights on what kind of setups are commonly used when analysing malware which served as a basis for the used environment. As inspired by Wagener et al. [4], a controlled environment was set up to ensure personal assets will not get compromised during the analysis while maintaining a real-world situation. Following is a list of the steps taken to analyse Zeus, these are described in more detail in the next sections.

- Set up a vulnerable Virtual Machine
- Install OllyDbg, Fiddler, InstallWatch Pro & Wireshark
- Analyse Zeus with OllyDbg by stepping through the system calls
- Analyse collected network traffic from client side with Wireshark
- Identify decryption step and extract key from memory

When the RC4 key has been extracted, the team will go a step further in looking at the behaviour of Zeus. In this phase a command and control server will be set up in a controlled local network along with a small set of infected clients.

3.1 Usage of a Virtual Machine & Fiddler

By running the Zeus executable inside a virtual machine with limited network settings, a safe and easily reproducible environment to test the malware is created. It is a safe environment because it provides the possibility to limit network access to other machines and because none of the changes made inside the virtual machine can have an effect on the operating system of the hosting computer. The system setup can be reproduced by taking snapshots during the process which allows reloading the system to a previously saved state to analyse other segments of the malware more specific when needed.

Since Zeus is malware creating a botnet, it requires its infected clients to have internet access in order to communicate with a Command and Control server. From this C&C server, it obtains configuration and execution instructions. As the botnet should not actually connect to a real server, internet access has been disabled in the virtual machine. To ensure the malware will behave regularly, Fiddler[3] is used

[3]http://www.telerik.com/fiddler

to mimic network behaviour. It intercepts HTTP requests and allows you to respond to those requests with preset answers. This fools the Zeus client into thinking that it just received a valid data stream from its C&C server, which can then be analysed further.

Following is a complete list of the different tools and resources that have been used during for the analysis:

- Windows 7
- VirtualBox
- OllyDbg
- Fakenet/Fiddler
- Wireshark
- InstallWatch Pro
- Zeus bot client exe (which was provided)
- Zeus C++ code [4]

4 Key Extraction

During the first phase of the analysis, the client malware was analysed in an attempt to find the encryption key used to encrypt communication with the C&C server. This phase includes setting up a safe and effective environment as well as performing a mix of both static and dynamic analysis to extract the RC4 key from the received binaries. The following sections will describe the process in finding the information in detail.

4.1 Process

Before running the executable in the test environment, the C++ code has been analysed to get a lead where to look for usages of the RC4 key. After inspecting the files describing the malware, the code responsible for the encryption key appeared to be located in the `crypt.cpp` file. With this knowledge the team was able to directly link the assembly instructions to the C++ code.

As the first step in the dynamic analysis, the malware execution has been performed with Install-Watch Pro on one client and OllyDBG and Fiddler on the other client. Installwatch Pro is a filesystem comparator tool and showed 20 files have been added, 21 files have been deleted, 6 files updated and many registry modifications. With this information, it became clear the executable copied itself to a different folder: `/users/<user_name>/AppData/Roaming/<variable_folder_name>/` . At the other client with Fiddler enabled it was also observed that the copied file requested the domain `hxxp://lifestyles.pp.ru` to download a file. This behaviour was noticed because of a pop-up from Windows 7 which notified the client an app needed network permission. From the request, it was easy to derive that a `back/config.bin` file was being retrieved from the server. At this point, the exact contents of this file are still unknown. By inspecting the corresponding C++ code, it was concluded the contents of the file are irrelevant for extracting the key since it was not part of the encryption process.

Using Fiddler, the client received a 200-OK response on the GET-request to `lifestyles.pp.ru`, and a fake `config.bin` file. Afterwards the Zeus binary in `AppData/Roaming/` was executed using OllyDbg. Memory breakpoints have been set before `WININET.InternetReadFile`; one on `0x4151BD` and one on `0x415267`. The placement of these breakpoints are based on the analysis of the C++ code for which a reduced call graph was created, which can be seen in Figure 2. The breakpoints are placed at the system calls where the client reads the encrypted data from its memory. The file is decrypted as soon as it is received, so that was the starting point to look for the RC4 key. After stepping through the code, a loop at `0x41918C` was observed. In this loop a byte was moved and another byte xor-ed, which was a good indication that this was the location of the RC4 algorithm. A breakpoint was set at the start of this probable RC4 function, and the experiment was executed again. By following the execution, the debugger stepped into a memory-copying function call at `0x4133F7`, indicating this should be the location where the RC4 key is copied before decrypting the received file. At `0x413406` the ESI points to the start of the key, which allowed the key to be extracted as shown in Figure 3.

[4] https://github.com/Visgean/Zeus/

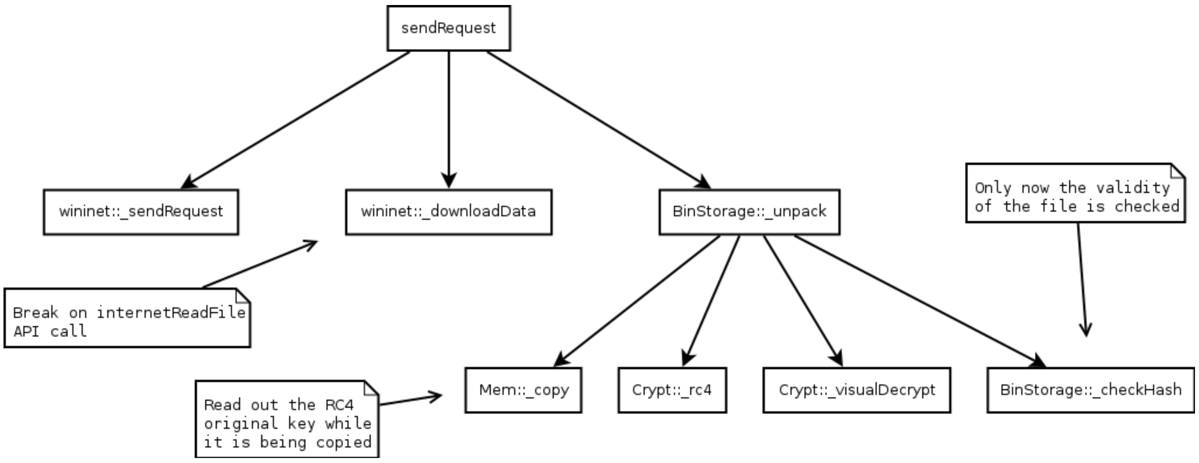
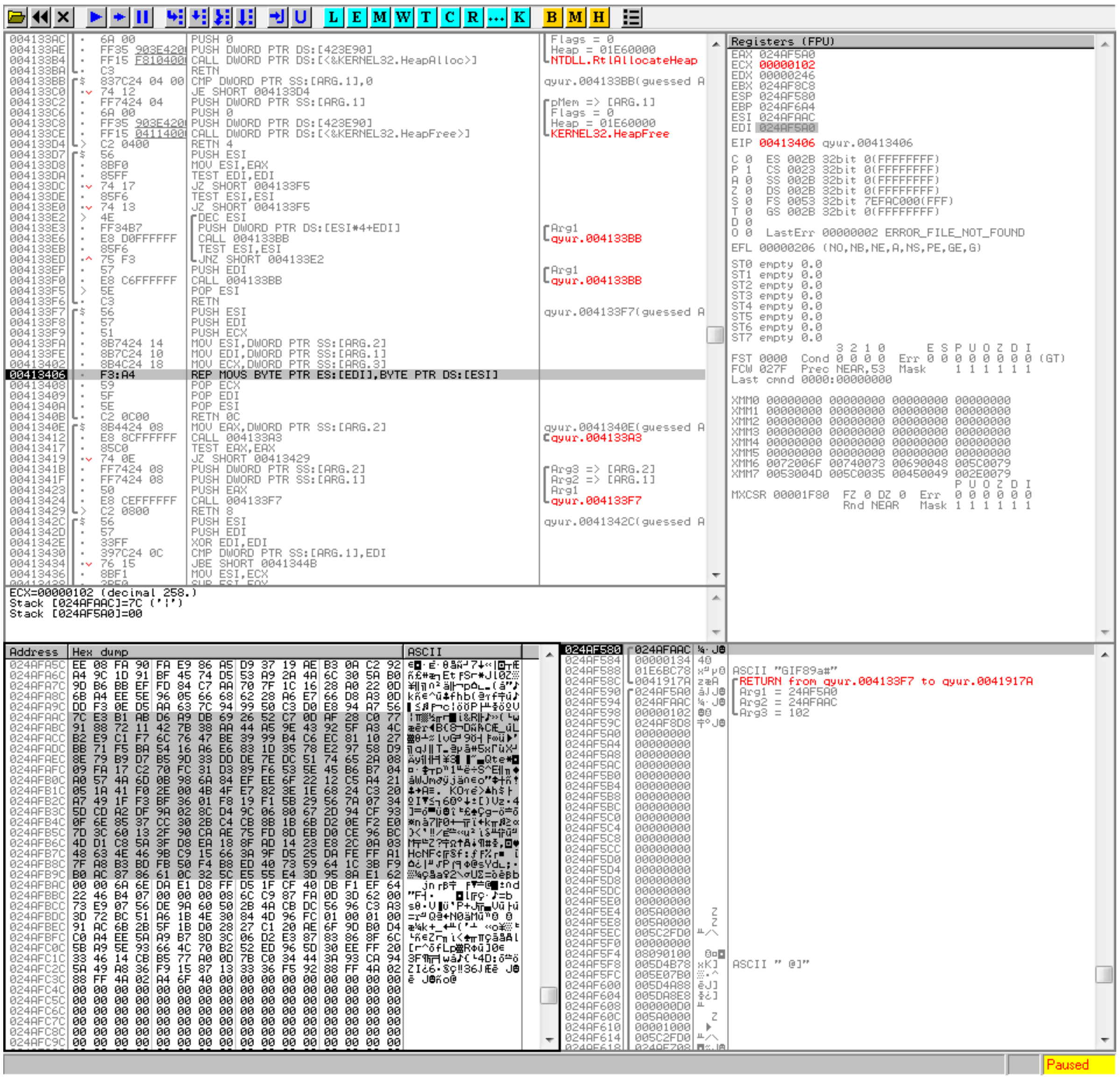


Figure 2: Overview of the call graph of Zeus. Calls are ordered chronologically from left to right.



```
004133AC  .  6A 00          PUSH 0                                        Flags = 0
004133AE  .  FF35 903E420   PUSH DWORD PTR DS:[423E90]                    Heap = 01E60000
004133B4  .  FF15 F810400   CALL DWORD PTR DS:[<&KERNEL32.HeapAlloc>]     NTDLL.RtlAllocateHeap
004133BA  .  C3             RETN
004133BB  $  837C24 04 00   CMP DWORD PTR SS:[ARG.1],0                    qyur.004133BB(guessed A
004133C0  .v 74 12          JE SHORT 004133D4
004133C2  .  FF7424 04      PUSH DWORD PTR SS:[ARG.1]                     pMem => [ARG.1]
004133C6  .  6A 00          PUSH 0                                        Flags = 0
004133C8  .  FF35 903E420   PUSH DWORD PTR DS:[423E90]                    Heap = 01E60000
004133CE  .  FF15 0411400   CALL DWORD PTR DS:[<&KERNEL32.HeapFree>]      KERNEL32.HeapFree
004133D4  >  C2 0400        RETN 4
004133D7  $  56             PUSH ESI
004133D8  .  8BF0           MOV ESI,EAX
004133DA  .  85FF           TEST EDI,EDI
004133DC  .v 74 17          JZ SHORT 004133F5
004133DE  .  85F6           TEST ESI,ESI
004133E0  .v 74 13          JZ SHORT 004133F5
004133E2  >  4E             DEC ESI
004133E3  .  FF34B7         PUSH DWORD PTR DS:[ESI*4+EDI]                 Arg1
004133E6  .  E8 D0FFFFFF    CALL 004133BB                                 qyur.004133BB
004133EB  .  85F6           TEST ESI,ESI
004133ED  .^ 75 F3          JNZ SHORT 004133E2
004133EF  .  57             PUSH EDI                                      Arg1
004133F0  .  E8 C6FFFFFF    CALL 004133BB                                 qyur.004133BB
004133F5  >  5E             POP ESI
004133F6  .  C3             RETN
004133F7  $  56             PUSH ESI                                      qyur.004133F7(guessed A
004133F8  .  57             PUSH EDI
004133F9  .  51             PUSH ECX
004133FA  .  8B7424 14      MOV ESI,DWORD PTR SS:[ARG.2]
004133FE  .  8B7C24 10      MOV EDI,DWORD PTR SS:[ARG.1]
00413402  .  8B4C24 18      MOV ECX,DWORD PTR SS:[ARG.3]
00413406  .  F3:A4          REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00413408  .  59             POP ECX
00413409  .  5F             POP EDI
0041340A  .  5E             POP ESI
0041340B  .  C2 0C00        RETN 0C
0041340E  $  8B4424 08      MOV EAX,DWORD PTR SS:[ARG.2]                  qyur.0041340E(guessed A
00413412  .  E8 8CFFFFFF    CALL 004133A3                                 qyur.004133A3
00413417  .  85C0           TEST EAX,EAX
00413419  .v 74 0E          JZ SHORT 00413429
0041341B  .  FF7424 08      PUSH DWORD PTR SS:[ARG.2]                     Arg3 => [ARG.2]
0041341F  .  FF7424 08      PUSH DWORD PTR SS:[ARG.1]                     Arg2 => [ARG.1]
00413423  .  50             PUSH EAX                                      Arg1
00413424  .  E8 CEFFFFFF    CALL 004133F7                                 qyur.004133F7
00413429  >  C2 0800        RETN 8
0041342C  $  56             PUSH ESI                                      qyur.0041342C(guessed A
0041342D  .  57             PUSH EDI
0041342E  .  33FF           XOR EDI,EDI
00413430  .  397C24 0C      CMP DWORD PTR SS:[ARG.1],EDI
00413434  .v 76 15          JBE SHORT 0041344B
00413436  .  8BF1           MOV ESI,ECX

ECX=00000102 (decimal 258.)
Stack [024AFAAC]=7C ('¦')
Stack [024AF5A0]=00
```

```
Registers (FPU)
EAX 024AF5A0
ECX 00000102
EDX 00000246
EBX 024AF8C8
ESP 024AF580
EBP 024AF6A4
ESI 024AFAAC
EDI 024AF5A0

EIP 00413406 qyur.00413406

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFAC000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr 00000002 ERROR_FILE_NOT_FOUND

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0 (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
Last cmnd 0000:00000000

XMM0 00000000 00000000 00000000 00000000
XMM1 00000000 00000000 00000000 00000000
XMM2 00000000 00000000 00000000 00000000
XMM3 00000000 00000000 00000000 00000000
XMM4 00000000 00000000 00000000 00000000
XMM5 00000000 00000000 00000000 00000000
XMM6 0072006F 00740073 00690048 005C0079
XMM7 0053004D 005C0035 00450049 002E0079
                                P U O Z D I
MXCSR 00001F80  FZ 0 DZ 0  Err  0 0 0 0 0 0
                Rnd NEAR   Mask 1 1 1 1 1 1
```

```
Address   Hex dump
024AFA5C  EE 08 FA 90 FA E9 86 A5 D9 37 19 AE B3 0A C2 92
024AFA6C  A4 9C 1D 91 BF 45 74 D5 53 A9 2A 4A 6C 30 5A B0
024AFA7C  9D B6 BB EF FD 84 C7 AA 70 7F 1C 16 28 A0 22 0D
024AFA8C  6B A4 EE 5E 96 05 66 68 62 28 A6 E7 66 D8 A3 0D
024AFA9C  DD F3 0E D5 AA 63 7C 94 99 50 C3 D0 E8 94 A7 56
024AFAAC  7C E3 B1 AB D6 A9 DB 69 26 52 C7 0D AF 28 C0 77
024AFABC  91 88 72 11 42 7B 38 AA 44 A5 9E 43 92 5F A3 4C
024AFACC  B2 E9 C1 F7 6C 76 47 BE 39 99 B4 C6 EC 81 10 27
024AFADC  BB 71 F5 BA 54 16 A6 E6 83 1D 35 78 E2 97 58 D9
024AFAEC  8E 79 B9 D7 B5 9D 33 DD DE 7E DC 51 74 65 2A 08
024AFAFC  09 FA 17 C2 70 FC 31 D3 89 F6 53 5E 45 B6 B7 04
024AFB0C  A0 57 4A 6D 0B 98 6A 84 EF EE 6F 22 12 C5 A4 21
024AFB1C  05 1A 41 F0 2E 00 4B 4F E7 82 3E 1E 68 24 C3 20
024AFB2C  A7 49 1F F3 BF 36 01 F8 19 F1 5B 29 56 7A 07 34
024AFB3C  5D CD A2 DF 9A 02 8C D4 9C 06 80 67 2D 94 CF 93
024AFB4C  0F 6E 85 37 CC 30 2B C4 CB 8B 1B 6B D2 0E F2 E0
024AFB5C  7D 3C 60 13 2F 90 CA AE 75 FD 8D EB D0 CE 96 BC
024AFB6C  4D D1 C8 5A 3F D8 EA 18 8F AD 14 23 E8 2C 0A 03
024AFB7C  48 63 4E 46 9B C9 15 66 3A 9F D5 25 DA FE FF A1
024AFB8C  7F A8 B3 BD FB 50 F4 B8 ED 40 73 59 64 1C 3B F9
024AFB9C  B0 AC 87 86 61 0C 32 5C E5 55 E4 3D 95 8A E1 62
024AFBAC  00 00 6A 6E DA E1 D8 FF D5 1F CF 40 DB F1 EF 64
024AFBBC  22 46 B4 07 00 00 00 08 6C C9 87 FA 0D 3D 62 00
024AFBCC  73 E9 07 56 DE 9A 60 50 2B 4A CB DC 56 96 C3 A3
024AFBDC  3D 72 BC 51 A6 1B 4E 30 84 4D 96 FC 01 00 01 00
024AFBEC  91 AC 6B 2B 5F 1B D0 28 27 C1 20 AE 6F 9D B0 D4
024AFBFC  C0 A4 EE 5A A9 B7 8D 3C 06 D2 E3 87 83 86 8F 6C
024AFC0C  5B A9 5E 93 66 4C 70 B2 52 ED 96 5D 30 EE FF 20
024AFC1C  33 46 14 CB B5 77 A0 0D 7B C0 34 44 3A 93 CA 94
024AFC2C  5A 49 A8 36 F9 15 87 13 33 36 F5 92 88 FF 4A 02
024AFC3C  88 FF 4A 02 A4 6F 40 00 00 00 00 00 00 00 00 00
024AFC4C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
024AFC5C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
024AFC6C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
024AFC7C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
024AFC8C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
024AFC9C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
024AF580  024AFAAC
024AF584  00000134
024AF588  01E6BC78  ASCII "GIF89a#"
024AF58C  0041917A  RETURN from qyur.004133F7 to qyur.0041917A
024AF590  024AF5A0  Arg1 = 24AF5A0
024AF594  024AFAAC  Arg2 = 24AFAAC
024AF598  00000102  Arg3 = 102
024AF59C  024AF8D8
024AF5A0  00000000
024AF5A4  00000000
024AF5A8  00000000
024AF5AC  00000000
024AF5B0  00000000
024AF5B4  00000000
024AF5B8  00000000
024AF5BC  00000000
024AF5C0  00000000
024AF5C4  00000000
024AF5C8  00000000
024AF5CC  00000000
024AF5D0  00000000
024AF5D4  00000000
024AF5D8  00000000
024AF5DC  00000000
024AF5E0  00000000
024AF5E4  005A0000
024AF5E8  005A0000
024AF5EC  005C2FD0
024AF5F0  00000000
024AF5F4  08090100
024AF5F8  005D4B78  ASCII " @J"
024AF5FC  005E07B0
024AF600  005D4A88
024AF604  005DA8E8
024AF608  000000D0
024AF60C  005A0000
024AF610  00001000
024AF614  005C2FD0

Paused
```

Figure 3: Extracting the RC4 key in OllyDBG

4.2 Key verification

In a full real-world setting where the client joins a botnet with an active C&C server, it would be feasible to intercept the encrypted messages from the C&C server. Since the RC4 key was extracted it should also be possible to identify the capabilities of Zeus from the client side by decrypting the intercepted messages. However, this would require an actual connection with the C&C server instead of just using the traffic generated by Fiddler. Besides the fact that it was not recommended to join an actual C&C server for this research project it was also impossible due to the unavailability of the server to which the clients was supposed to connect with as it was already offline.

Due to the limited time for this project, it was unfeasible to verify the extracted key in a more scientific manner. However, as the received sample has been analysed before by other researchers it was confirmed the extracted key was correct.

5 Creating and controlling a Zeus Botnet

The next assignment was to set up a private botnet to control a client that has been infected with Zeus. This requires the configuration and setup of a controlled internal network with at least two VM's. One client will be the infected client and thus acting as a bot and the other will function as the Command and Control server which will control the bot as soon as it joined.

5.1 Network Configurations

Setting up a botnet over any public network poses significant risks for other connected clients. However, a botnet requires a functioning network to operate. For this reason it was decided to create a dedicated private network for the analysis. Two virtual machines have been created which are connected by an internal network configured with Oracle VirtualBox. One VM was running Debian and should function as the C&C server. The second VM was running Windows 7, which would function as the infected client. Both clients have to be assigned with a statically configured IP address in order to enable communication between the two.

5.2 C&C server

In the analysed Zeus source code, a folder called `output` also contains the files needed to set up the Command and Control server, as well as makefiles for the bot. These samples will be used for setting up the C&C server and to infect the client, in order to create a fully functioning botnet.

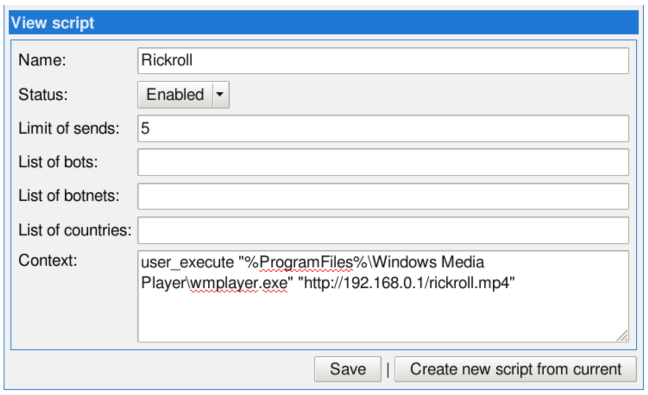
The `server[php]` folder contains all the files needed to set up a functioning C&C server. When serving this folder using webserver software (such as Apache), installation can be completed by browsing to the `/install` folder.

The control panel can be configured by providing a username for the root user (and optionally a password), the information for the MySQL server and a local folder to which the reports will be saved. After a successful install, it's possible to log in and take control over your very own Zeus powered botnet.

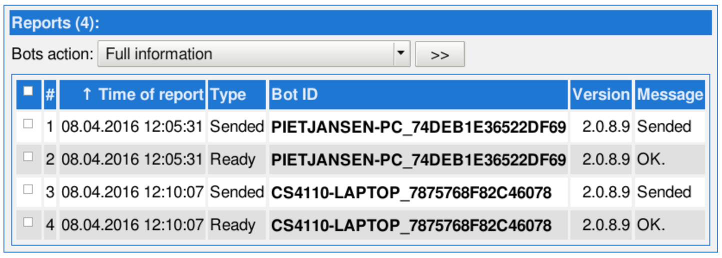
5.3 Bot configuration

The README file in the Zeus code repository provides a surprisingly detailed guide to generate a bot. It describes how the config file in `output/builder/config.txt` should be setup, which configurations are statically added to the executable of the bot and which configurations are dynamic which should be downloaded from the server. The `config.txt` file is used by the bot builder to create the executable of the bot. This embeds the RC4 key into the client by setting the `encryption_key` which corresponds to the key used at the control panel when the server side was installed. The config file should also specify the location of the `config.bin` and what URL the Command and Control server is located.

The last step was to compile the bot. The cloned code from github also specifies a compiler interface. The interface asks for the source of the configuration file and also enables the user to set some specific options. By clicking *Build the bot executable*, the `zeus-bot.exe` is created and ready to be executed.



Name:	Rickroll
Status:	Enabled
Limit of sends:	5
List of bots:	
List of botnets:	
List of countries:	
Context:	user_execute "%ProgramFiles%\Windows Media Player\wmplayer.exe" "http://192.168.0.1/rickroll.mp4"



	#	↑ Time of report	Type	Bot ID	Version	Message
☐	1	08.04.2016 12:05:31	Sended	**PIETJANSEN-PC_74DEB1E36522DF69**	2.0.8.9	Sended
☐	2	08.04.2016 12:05:31	Ready	**PIETJANSEN-PC_74DEB1E36522DF69**	2.0.8.9	OK.
☐	3	08.04.2016 12:10:07	Sended	**CS4110-LAPTOP_7875768F82C46078**	2.0.8.9	Sended
☐	4	08.04.2016 12:10:07	Ready	**CS4110-LAPTOP_7875768F82C46078**	2.0.8.9	OK.

Figure 4: The parameters and results for executing a script.

5.4 C&C server functionality

The command and control server exposes all the capabilities of the bot to the bot maintainer. The control panel allows the bot herder to gather information about the clients who have joined the botnet. In addition to that, it can display aggregated statistics about software versions of the bot/operating system and the amount of bots currently connected to the C&C server.

The maintainer is also able to write scripts for the bots to execute. These scripts will be executed as soon as the bot connects to the C&C server. The scripting system allows filters to target specific manually selected bots or bots in a certain country. A graphical representation of configuring a script is given in Figure 4. Each bot will periodically check the C&C server for updates and queued commands. The C&C server keeps track of what bot has executed what script, to prevent duplicate script execution. The scripting language has some standard functionality but the most important one is the ability to execute arbitrary commands with the rights of the current user. The result of executing the script in Figure 4 is shown in Figure 5. This basically gives the botnet maintainer unrestricted access and full control over the infected client.

The scripts the bot can process are simple commands, no control flow is implemented. The commands available can be summarised with:

- Operating system shutdown/reboot
- Bot uninstall/update
- Backconnect add/remove
  The backconnect functionality allows the bot to start a remote login or proxy server (RDP, VNC or Socks) that the bot maintainer can use.
- HTTPInject enable/disable
  HTTPInject refers to a functionality where the bot hooks into the operating system calls to create HTTP connections. For specific websites (such as ING.nl), it can inject extra HTML or javascript code. The leaked source code contains a lot of examples that allows to steal banking-related data from the user.
- User remove/log-off
- Execute command

- URL block/unblock
  Stops certain URL's from being loaded via the hooks into the OS HTTP functions. The error returned is not reachable, so the user is unaware why these websites don't work.
- Certificates get/remove
- Cookies get/remove
  Retrieve cookies from the browser and adobe Flash player.
- Set homepage
- FTP/Email get details
  There are built-in lists of well known FTP and email clients and where they store the credentials, which can be retrieved with this command.

6 Observations

During the project several interesting observations were made, which gave us insights into how the many different capabilities are actually working. Some of the observations have already been discussed as it was used during the process of identifying the RC4 key, or setting up the botnet. However, other interesting observations which might be useful for detecting or preventing infections like these will be described in this section. This section will describe the most interesting observations from the client's perspective.

During the first phase of the project, a client was infected while using the filesystem comparator tool InstallWatch Pro which was described before. When inspecting the results from InstallWatch Pro, it was observed that next to the addition of expected files to install Zeus, all the cookies from Internet Explorer were deleted, as shown in Figure 6. While the reason for removing the cookies during installation might not look relevant at first sight, it reveals a very important reason in the success of Zeus. Because Zeus is designed to collect all user information, there should be some method to force an infected user to resubmit its valid credentials. By removing the cookies of the web browser, a user is forced to resubmit its valid credentials after Zeus has been successfully installed. This strategy allows the malware to collect as most valid accounts and passwords as possible.

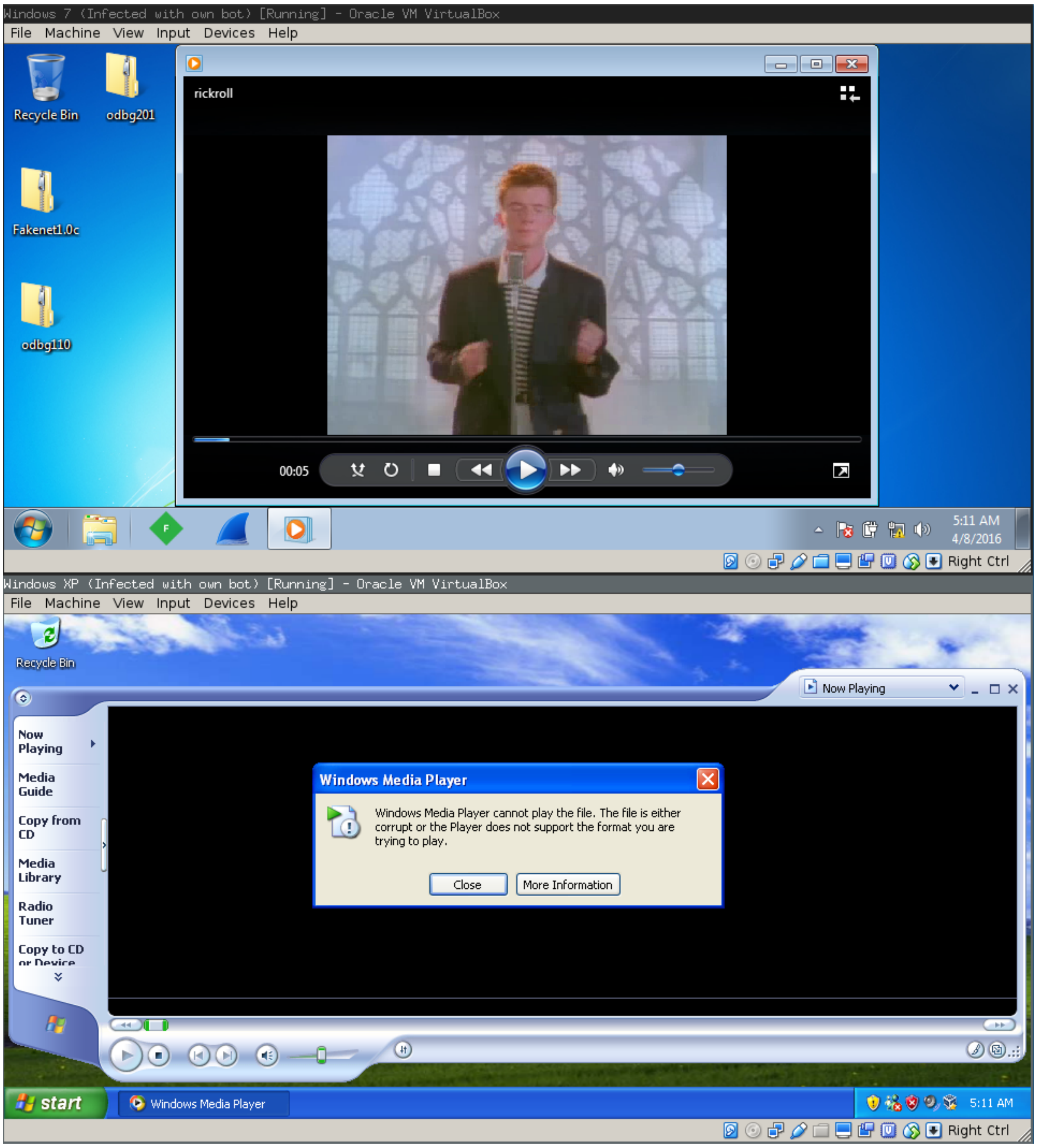
As soon as the client has been infected, it immediately sends a GET-request to retrieve the config file from its C&C server. The server would reply with a 200-OK status code, and return the encrypted configuration file. This is the first file received by the bot after infection and it is decrypted by using the RC4 key. Interestingly, all traffic between the bot and C&C was encrypted except when a screenshot is being transferred. It remains unknown why this traffic in specific is not encrypted. Some sources claim[3] that a Zeus bot opens a port to which the C&C server can listen to. This connection is primarily used for receiving screenshots when requested almost instantaneously.

After the client has received the config file, it is observed the client sends a set of HTTP POST requests without receiving any response from the server. Other network traffic also included for example a few attempts to visit Google which was most likely performed to verify the connectivity to the internet.

Characterising observations in the network traffic like these could support the detection of a possible Zeus infection. For example, by writing rules based on these observations for an Intrusion Detection System could improve the detection rate of Zeus.

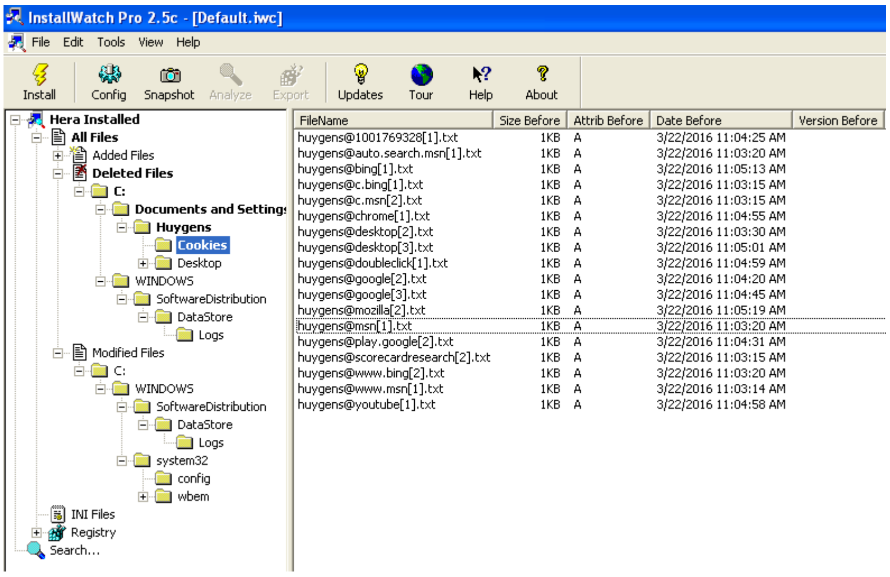
7 Conclusion & Future work

When working on first part of the coursework, we choose to focus our selected papers on botnets. We hoped that our labwork would have something to do with a botnet and we were in for a treat! With the Zeus botnet client program provided we were able to extract the RC4 and set up a internal network containing a Command and Control server and a bot. The key extraction used quite a bit of skill with both static and dynamic analysis. The static analysis was done by inspecting the C++ code that was found on Github and the dynamic analysis was done via running and stepping through the assembly code of the executable. During the extraction of the key we learned that the extraction can be quite difficult, and recognise the potential problems obfuscation can cause. We reckon that the extraction



Windows Media Player cannot play the file. The file is either corrupt or the Player does not support the format you are trying to play.

Figure 5: After executing the script



FileName	Size Before	Attrib Before	Date Before	Version Before
huygens@1001769328[1].txt	1KB	A	3/22/2016 11:04:25 AM	
huygens@auto.search.msn[1].txt	1KB	A	3/22/2016 11:03:20 AM	
huygens@bing[1].txt	1KB	A	3/22/2016 11:05:13 AM	
huygens@c.bing[1].txt	1KB	A	3/22/2016 11:03:15 AM	
huygens@c.msn[2].txt	1KB	A	3/22/2016 11:03:15 AM	
huygens@chrome[1].txt	1KB	A	3/22/2016 11:04:55 AM	
huygens@desktop[2].txt	1KB	A	3/22/2016 11:03:30 AM	
huygens@desktop[3].txt	1KB	A	3/22/2016 11:05:01 AM	
huygens@doubleclick[1].txt	1KB	A	3/22/2016 11:04:59 AM	
huygens@google[2].txt	1KB	A	3/22/2016 11:04:20 AM	
huygens@google[3].txt	1KB	A	3/22/2016 11:04:45 AM	
huygens@mozilla[2].txt	1KB	A	3/22/2016 11:05:19 AM	
huygens@msn[1].txt	1KB	A	3/22/2016 11:03:20 AM	
huygens@play.google[2].txt	1KB	A	3/22/2016 11:04:31 AM	
huygens@scorecardresearch[2].txt	1KB	A	3/22/2016 11:03:15 AM	
huygens@www.bing[2].txt	1KB	A	3/22/2016 11:03:20 AM	
huygens@www.msn[1].txt	1KB	A	3/22/2016 11:03:14 AM	
huygens@youtube[1].txt	1KB	A	3/22/2016 11:04:58 AM	

Figure 6: Removed Internet Explorer Cookies

can be even more troublesome when the C++ code was not available. After succeeding in extracting the key, the following step was to set up a botnet using virtual machines. Due to the ease of use with the practical user interface, it was noticed that even without much knowledge of the malware itself it was relatively easy to set up a fully functioning botnet.

It took some time to get right because of our unfamiliarity and added configuration due to usage of virtual machines and an internal network. When done by someone with more experience in setting this up, it can be done quite quickly. This shows that the creators of botnets, and probably malware in general, are skilled in creating good working programs focused on their intended goal. After setting up the botnet, we tried out what kind of access features the command and control server provides. The command and control server provides for near real time screenshot footage of the bot and provides for the possibility of running scripts directly on the bot. We recognise this as a huge threat to the security of an average computer user. An average computer user will not notice the bot being present on the system after installation. During installation, the network access the bot needs to download his configuration file makes Windows 7 launch a pop-up warning about this. When 'no' is selected this will disable the functionality of the bot, but when answered with 'yes' gives full access to the infected computer. We reckon that many people automatically answer yes to these pop-ups just to make them disappear. After installation no other pop-ups or warnings are shown by the operating system. Although this practical gave us some great insight into how botnets operate some aspects remain open for further exploration. One of the interesting aspects to look into, is the network traffic that is generated by the bot and the command and control server. The information about the network traffic could give possibilities to detect a botnet. A second aspect that can be interesting is setting up multiple bots, to test out the interaction with the command and control server and network traffic on a bigger scale.

References

[1] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 621–634. ACM, 2009.

[2] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2):6, 2012.

[3] I. Incorporated. Reversal and analysis of zeus and spyeye banking trojans. Technical report, IOActive Inforporated, 2012.

[4] G. Wagener, A. Dulaunoy, et al. Malware behaviour analysis. *Journal in computer virology*, 4(4):279–287, 2008.