

Financial Malware Analysis- Secrets of ZeuS

Rushita Dave¹, Anisetti Anjaneyulu²

¹Forensics Analyst eSF Labs Ltd.

Abstract- Malware becomes one of the internet's major security threats these days. Malware is the most widespread threat towards IT now. Because of the huge quantity of new malware samples, researchers rely on dynamic malware analysis. Malware analysis is an important part of understanding the objectives of the malware and how to defend against this threat. To be able to defend against the threat imposed by malware we need to understand both how and why the malware exists.

I. INTRODUCTION

Malware is a general term for a piece of software inserted into an information system to cause harm to the system. Malware can gain remote access to system, record and send data to a third party without the user's permission or knowledge. Malware are commonly described as viruses, worms, trojan horses, backdoors, keystroke loggers, rootkits or spyware. Software may contain vulnerabilities, in its structure caused by imperfect coding. Once this type of vulnerabilities is revealed, malware can be developed to exploit them for malicious purposes.

Zeus refers to an entire family of trojans and their respective botnets. Zeus installs a rootkit component to stay hidden on infected systems. To bypass firewalls and to remain active on infected systems, Zeus inserts itself in the address space of other running processes typically Windows Explorer.

Features of Zeus: Capture credentials over HTTP and HTTPS, Steals HTTP and flash cookies, encrypted configuration file, modifies local host files, unique bot identification string.

II. MEMORY FORENSICS FOR MALWARE ANALYSIS

RAM contains critical information about the runtime state of the system while the system is active. By capturing an entire RAM and analyzing it on a separate computer, it is possible to reconstruct the state of the original system, including what applications were running, which files those applications were accessing, which network connections were active, and many other artifacts. For these reasons, memory forensics is extremely important to incident response. Memory forensics helps with unpacking, rootkit detection, and reverse engineering.

A. Tools to capture memory

ManTech Memory DD

Mandiant Memoryze

FTK Imager

DumpIt

Virtual Machine Files (vmem):

Virtual machines provide a useful environment for dynamic analysis of malware. Acquire RAM from guest machines by just suspending or pausing the VM, and guest's RAM will be written to a file on the host's disk. Then grab the ".vmem" file for analysis. The default location of the vmem file for VMware is %My Documents%\My Virtual Machines\<VM Name>*.vmem

B. Tool to analyze the memory

Volatility

The Volatility Framework is an open collection of tools for the extraction of digital artifacts from volatile memory samples. Whether your memory dump is in raw format, a Microsoft crash dump, hibernation file, or virtual machine snapshot, Volatility is able to work with it. It is a very good tool used for malware analysis as malware runs in the memory.

III. MALWARE ANALYSIS USING VOLATILITY

A. Image information (imageinfo)

```
python vol.py -f Zeus_Final.vmem imageinfo
```

Set a profile to tell Volatility what operating system the dump came from. The "imageinfo" output suggests profile that should be passed as the parameter to --profile=PROFILE

B. Scan for the ZeuS Malware (zbotscan)

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem zbotscan
```

This will show the version of the ZeuS - ZEUS 2.1.0.1, malware infected process with pid - 1512, registry key - HKCU\SOFTWARE\Microsoft\Irpao and executable - ixmo.exe

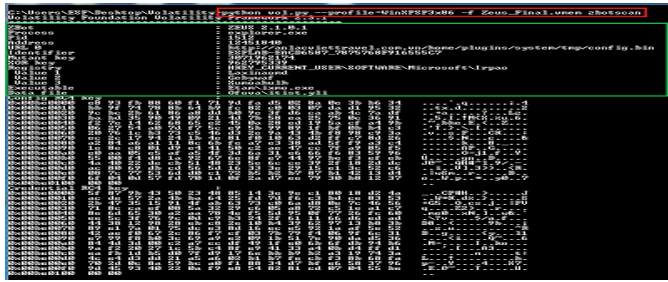


Fig 1. zbot command to check for the infected process

C. Checking the process list (psscan)

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem psscan
```

This is used for analysis of malware and rootkit activities. It scans for inactive, hidden and unlinked processes by a malware or rootkit.

Check for the process that was shown in the zbotscan command.

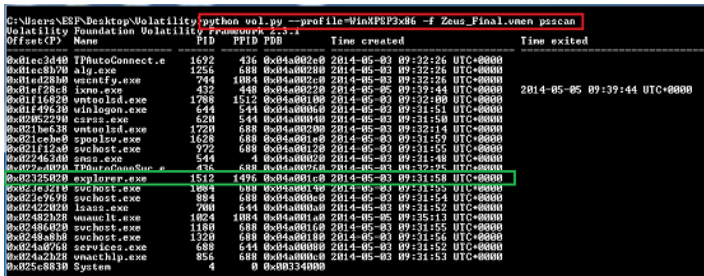


Fig.2 List of the process running in the system at the time the dump was taken

Here it shows the detail about the "explorer.exe" which was found before.

D. Handles of the Process (handles)

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem handles -p 1512 -t Process
```

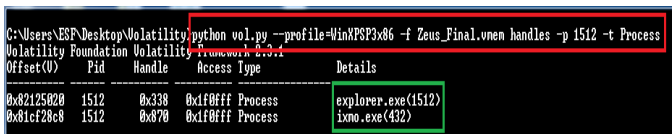


Fig.3 Process handles of the "explorer.exe"

Here it shows that explorer.exe is communicating with 2 processes. One is itself explorer.exe and other is ixmo.exe which is executable shown before.

E. Check for the hooks (apihooks)

It is used to find API hooks in user mode or kernel mode. This finds IAT, EAT, Inline style hooks. For Inline hooks, it detects CALLs and JMPs to direct and indirect locations.

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem handles
```

It show the hook type - Inline hook, Process - explorer.exe, victim module - ntdll.dll, kernel32.dll, user32.dll.

F. Check for the modules (ldrmodules)

It is also possible for malware to hide a DLL by simply overwriting the path. Tools that only look for unlinked entries may miss the fact that malware overwrite C:\malicious.dll to show C:\windows\system32\kernel32.dll.

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem ldrmodules
```

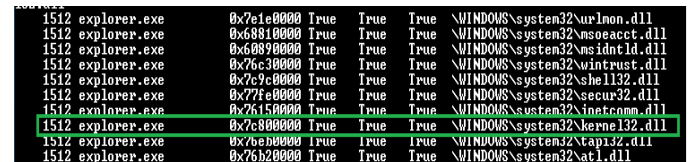


Fig.4 modules of explorer.exe

It shows the base address of process that can be further use for analyzing the process for finding the injected code in it.

G. Volatility Shell (volshell)

Volshell is the terminal of the volatility itself.

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem volshell
```

```
cc(pid=1512)
```

```
db(offset)
```

offset - found from the result of the ldrmodules.

```
C:\Users\ESP\Desktop>volatility.py python vol.py --profile=WinXPSP3x86 -f Zeus_Final.vmem volshell
Volatility Foundation Volatility Framework 2.3.1
Current context: process System, pid=4, ppid=0 DTB=0x334000
Welcome to volshell! Current memory image is:
File: C:\Users\ESP\Desktop\Volatility\Zeus_Final.vmem
File type: vmem
File size: 1073896
File md5: 1b4c11b4c11b4c11b4c11b4c11b4c11b
>>> cc(pid=1512)
Current context: process explorer.exe, pid=1512, ppid=1496 DTB=0x4a001c0
>>> db(0x77c00000)
0x77c00000 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00
0x77c00010 18 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
0x77c00020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x77c00030 00 00 00 00 00 00 00 00 00 00 00 00 d8 00 00 00
0x77c00040 0e 1f ha 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68
0x77c00050 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f
0x77c00060 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20
0x77c00070 6d 6f 64 65 2e 0d 00 0a 24 00 00 00 00 00 00 00
>>>
```

Fig.5 volatility shell

H. Finding malicious code (malfind)

The malfind is used to find hidden or injected code/DLLs in user mode memory. It is also used to locate sequence of bytes, regular expressions, ANSI strings, or Unicode strings in user mode or kernel memory. The purpose of malfind is to locate DLLs that standard methods/tools do not see.

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem malfind
```

```
Process: explorer.exe Pid: 1512 Address: 0xc10000
User Tag: Uuids Protections: EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6
0x00c10000 b8 35 00 00 00 e9 8b d1 cf 7b 68 6e c2 00 00 e9 .5.....Chl...
0x00c10010 94 63 a0 7b 8b ff 55 8b ec e9 6e 11 e0 7b 8b ff .e.C-.U...l..c.
0x00c10020 55 8b ec e9 92 2e 60 76 8b ff 55 8b ec e9 74 60 U...U...t...t.
0x00c10030 5b 76 8b ff 55 8b ec e9 8a e9 5b 76 8b ff 55 8b l.v...U...lv..U.
0xc10000 m35000000 MOV EBX, 0x35
0xc10005 e98bd1cf7b JMP 0x7c90a195
0xc1000a 686e020000 PUSH DWORD 0x26c
0xc1000f e99463d07b JMP 0x7c9163a8
0xc10014 8bff MOV EDI, EDI
0xc10016 55 PUSH EBP
0xc10017 8bec MOV EBP, ESP
0xc10019 e96c11c07b JMP 0x7c81118a
0xc1001e 8bff MOV EDI, EDI
0xc10020 55 PUSH EBP
0xc10021 8bec MOV EBP, ESP
0xc10023 e9922e6076 JMP 0x77212ec1
0xc10028 8bff MOV EDI, EDI
0xc1002a 55 PUSH EBP
0xc1002b 8bec MOV EBP, ESP
0xc1002d e974605b76 JMP 0x771c60a6
0xc10032 8bff MOV EDI, EDI
0xc10034 55 PUSH EBP
0xc10035 8bec MOV EBP, ESP
0xc10037 e98ae95b76 JMP 0x771ce966
0xc1003c 8bff MOV EDI, EDI
0xc1003e 55 PUSH EBP
0xc1003f 8b DB 0x8b
```

Fig.6 malfind command for hidden or injected code

detect the presence of Zeus. The first memory segment was detected because its executable and has meaning that there is memory mapped file already occupying the space.

I. Yara File Scan (yarascan)

Yara file is a set of rules to identify the behavior of the malware. It contains the list of the strings that is needed to be searched in the memory image for analyzing the behavior of the malware.

```
C:\Users\ESP\Desktop\Volatility>python vol.py --profile=WinXPSP3x86 -f Zeus_Final.vmem paroscan -y "C:\Users\ESP\Desktop\Volatility\yara rules\zeus_vim_01\Zeus_des_
Volatility Foundation Volatility Framework 2.3.1
Rule: zeus_vim_01\Zeus_des
Author: Process explorer.exe Pid 1512
0x00014000 69 74 74 70 7a 72 71 77 77 2e 49 5f 67 6c http://www.google
0x00014010 45 2e 63 6f 44 2f 77 45 52 68 70 00 02 00 63 00 e.com/webp/h.c.
0x00014020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x00014030 24 00 66 00 00 00 00 73 61 63 63 73 00 00 00 .....ocbr...
Rule: zeus_vim_01\Zeus_des
Author: Process explorer.exe Pid 1512
0x00017000 58 5f 5f 4f 70 65 6e 54 43 58 53 6f 63 65 74 PR_OpenTCPsocket
0x00017010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...PR_Close...PR
0x00017020 58 5f 5f 4f 64 00 58 5f 5f 5f 5f 5f 5f 5f 5f 5f PR_ReadPRWrite
0x00017030 00 00 00 52 46 42 20 30 30 33 2e 30 30 33 3a .....RFP.003.003.
Rule: zeus_vim_01\Zeus_des
Author: Process explorer.exe Pid 1512
0x00017200 39 5f 5f 4f 6f 73 65 00 00 00 00 00 53 5f 5f PR_Close...PR_R
0x00017210 65 61 64 00 58 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f PR_ReadPRWrite
0x00017220 52 46 42 20 30 33 2e 30 30 33 3a 00 00 00 00 RFP.003.003....
0x00017230 5f 46 42 20 00 00 00 53 00 79 00 73 00 46 00 RFP.....S.S.L.L.
Rule: zeus_vim_01\Zeus_des
Author: Process explorer.exe Pid 1512
0x00017400 00 00 00 52 46 42 20 30 33 2e 30 30 33 3a .....RFP.003.003.
0x00017410 00 00 00 52 46 42 20 30 33 2e 30 30 33 3a .....RFP.....S.S.
0x00017420 00 00 00 52 46 42 20 30 33 2e 30 30 33 3a .....RFP.....S.S.
0x00017430 73 00 46 00 69 00 73 00 79 00 55 00 69 00 65 00 t.v.l.c.l.c.l.c.
Rule: zeus_vim_01\Zeus_des
Author: Process explorer.exe Pid 1512
0x00017600 52 46 42 20 30 33 2e 30 30 33 3a 00 00 00 00 RFP.003.003....
0x00017610 5f 46 42 20 00 00 00 53 00 79 00 73 00 46 00 RFP.....S.S.L.L.
0x00017620 52 00 46 00 69 00 73 00 79 00 55 00 69 00 65 00 t.v.l.c.l.c.l.c.
0x00017630 32 00 00 00 40 00 40 00 43 00 6e 00 69 00 .....N.L.L.L.L.L.
```

Fig.7 volatility shell

J. Printing registry key (printkey)

This command is display the subkeys, values, data, and data types contained within a specified registry key, use the printkey command.

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem printkey -K "Software\Microsoft\Irpao"
```

```
C:\Users\ESP\Desktop\Volatility>python vol.py --profile=WinXPSP3x86 -f Zeus_Final.vmem printkey -K "Software\Microsoft\Irpao"
Volatility Foundation Volatility Framework 2.3.1
Legend: ($) = Stable (V) = Volatile
Registry: \Device\Harddisk0\Volume1\Documents and Settings\Administrator\NTUSER.DAT
Key name: Irpao ($)
Last updated: 2014-05-05 09:39:44 UTC+0000
Subkeys:
Values:
REG_BINARY Xmosulh1 : ($)
0x00000000 e9 46 6b 21 01 0e 0d ab a7 79 ab 60 0b 27 89 .LPH.....y..'.
0x00000010 f6 64 59 31 00 1a fa fb hd d5 72 49 20 53 ea 68 ..d1.....p.f.s.h
0x00000020 ec 34 50 9c 72 06 2e fb 2e 79 5a fb 2f bc e0 24 ..P.s.....p.f.s.h
0x00000030 ab 2e c0 e5 2d e1 ea e3 1b 5f 3f 16 54 53 60 c5.....t..f..
0x00000040 54 57 7d e4 23 30 1c 82 af 74 1e 44 8a e4 fe aa 1D...H...t..h...
0x00000050 dc 40 c6 c3 98 97 09 53 6e 22 74 85 ac 84 41 22 d.....S"t...h"
0x00000060 73 6f fa 6a 51 40 hd 56 77 92 32 2a e4 65 00 f3 au..QH.Uw.2..e..
0x00000070 e3 00 h3 h3 .....
```

Fig.8 volatility shell

b. "ControlSet001\Services\SharedAccess\Parameters\Firewall Policy\StandardProfile"

```
C:\Users\ESP\Desktop\Volatility>python vol.py --profile=WinXPSP3x86 -f Zeus_Final.vmem printkey -K "ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile"
Volatility Foundation Volatility Framework 2.3.1
Legend: ($) = Stable (V) = Volatile
Registry: \Device\Harddisk0\Volume1\WINDOWS\system32\config\system
Key name: StandardProfile ($)
Last updated: 2014-05-03 09:50:29 UTC+0000
Subkeys:
Values:
REG_DWORD EnableFirewall : ($) 0
```

It is used to detect if the windows firewall is enabled or disabled.

K. Checking event logs (evtlogs)

```
python vol.py --profile=WinXPSP2x86 -f Zeus_2_Final.vmem  
evtlogs -D C:\Users\ESF\Desktop\Dump
```

The "sysevent.txt" file is the important file to be analyzed from the four files that is acquired by this command.

It shows that a socket operation was attempted to unreachable host. It means that some process of the system has tried to connect to the remote server. It also shows that the computer browser is stopped. It means that at some point of time the computer browser was stared.

VI. CONCLUSION

Volatility tool gives the detailed result of the malware behavior. It shows the infected process among the all running processes in the system, the malware information, the infected executable. It also shows the string of the malware. It states that the malware tries to hide itself behind the legitimate process of the system and infect the system.

REFERENCES

- [1] Malware Analyst's cookbook and DVD: Tools and Techniques for Fighting Malicious Code Paperback by Michael Ligh (Author), Steven Adair (Author), Blake Hartstein (Author), Matthew Richard (Author)
- [2] Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software Paperback by Michael Sikorski (Author), Andrew Honig (Author)
- [3] Mandiant Practical Malware Analysis by Kris Kendall
- [4] https://code.google.com/p/volatility/wiki/CommandReference#Image_Identification
- [5] <https://code.google.com/p/volatility/wiki/LinuxMemoryForensics>
- [6] <http://msdn.microsoft.com/en/US>
- [7] Symantec : ZeusS: King of the bots by Nicolas Falliere and Eric Chien
- [8] www.secarma.co.uk/blog/2013/04/03/how-to-installing-volatility-on-windows-7-64-bit/
- [9] SANS CheetSheet v2.3
- [10] Reversing and Malware Analysis Training [2012]
- [11] Malware Analysis: An Introduction, SANS Institute InfoSec Reading Room, by Author: Dennis Distler