**Memory forensics: Using Volatility Toolkit to extract malware samples from memory dump**

29 May 2013 | Filed under: [Malware analysis](http://kromer.pl/category/malware-analysis/) and tagged with: [Dump](http://kromer.pl/tag/dump/), [DumpIt](http://kromer.pl/tag/dumpit/), [Forensics](http://kromer.pl/tag/forensics/), [mdd](http://kromer.pl/tag/mdd/), [Memory](http://kromer.pl/tag/memory/), [Metasploit](http://kromer.pl/tag/metasploit/), [psexec](http://kromer.pl/tag/psexec/), [Volatility](http://kromer.pl/tag/volatility/), [ZBot](http://kromer.pl/tag/zbot/)

Sometimes it may happen that you see some signs of malware on one of your Windows hosts,  like f. e. IDS detecting traffic to a known C&C server or some common pattern, but installed AntiVirus doesn’t raise any alarm. You can, of course, RDP to that host or access it directly and begin investigation, but before you’ll find out what’s going on, malware can remove itself and any traces of it’s activity. Here’s an example scenario of collecting evidence from computer memory.

**Creating memory dump**

First you’ll need a memory dump. Even if malware will remove itself after that, you’ll still be able – in most cases – to get the sample of it and perform analysis. There is a variety of tools which can be used to dump memory into the file, the one I like the most is [MoonSols DumpIt](http://www.moonsols.com/2011/07/18/moonsols-dumpit-goes-mainstream/" \o "MoonSols DumpIt" \t "_blank). It’s fast and easy to use. Unfortunately when it comes to remote execution, it becomes problematic. There’s no way to avoid to confirm dump creation remotely. For remote execution try [mdd](http://sourceforge.net/projects/mdd/" \o "mdd on Sourceforge" \t "_blank) instead. Combine it with [PsExec](http://technet.microsoft.com/en-us/sysinternals/bb897553.aspx" \o "Sysinternals: PsExec" \t "_blank) (if you’re running Windows), which is part of Sysinternals PsTools and if you’re a Linux user, you can use psexec module in [Metasploit Framework](http://www.metasploit.com/" \o "Metasploit home page" \t "_blank) or [smb-psexec.nse](http://nmap.org/nsedoc/scripts/smb-psexec.html" \o "Nmap: smb-psexec.nse script" \t "_blank) script for [Nmap](http://nmap.org/" \o "Nmap home page" \t "_blank). If you have direct access to investigated machine, run DumpIt from pen drive, network share or simply [Google for DumpIt](http://www.google.com/search?q=DumpIt), the first result is the right one. DumpIt requires administrator privileges – Windows 7 will most likely ask you to escalate prvileges, on XP you’ll have Shift + Right Click and select ‘Run As…’. If you’re on Windows and want to run mdd remotely, place the PsExec.exe and mdd\_1.3.exe (or any other version) in the same directory, easily accessible from command line. Remember that username and password will be sent over network in plain text, so it’s very easy to capture it by an unauthorized person. To create a memory dump, run the following command:

**C:\Forensic>**PsExec.exe \\HOSTNAME\_OR\_IP -u DOMAIN\privileged\_account -p passwd -c mdd\_1.3.exe - -o C:\MEMORY.DMP

On Linux you can use Metasploit psexec module. First start Metasploit (Gentoo doesn’t create symlinks without msf version):

**$** msfconsole4.4

Now launch the psexec module and set logon details:

**msf >** use exploit/windows/smb/psexec

**msf exploit(psexec) >** set RHOST HOSTNAME\_OR\_IP

RHOST => HOSTNAME\_OR\_IP

**msf exploit(psexec) >** set LHOST YOUR\_HOSTNAME\_OR\_IP

LHOST => YOUR\_HOSTNAME\_OR\_IP

**msf exploit(psexec) >** set SMBDomain DOMAIN

SMBDomain => DOMAIN

**msf exploit(psexec) >** set SMBUser privileged\_user

SMBUser => privileged\_user

**msf exploit(psexec) >** set SMBPass password

SMBPass => password

**msf\_exploit(psexec) >** set VERBOSE yes

VERBOSE => yes

Now it gets a little tricky. Usually I have a file share with mdd executable accessible from investigated machine. In different environment you can create a file share with mdd on your machine and access it from remote host. It’s time to connect:

**msf exploit(psexec) >** exploit

Now we’ll get the shell and run mdd:

**meterpreter >** shell

Process 4324 created.

Channel 1 created.

Microsoft Windows [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp.

**C:\WINDOWS\system32>** \\SERVER\share\mdd\_1.3.exe -o C:\MEMORY.DMP

When memory dump is ready you can fetch the file from \\HOSTNAME\_OR\_IP\C$\MEMORY.DMP using Windows Explorer (remember to access it as privileged user) or using smbclient on Linux or any other method which is accessible and preferred by you.

**Analyzing memory dump**

Time for the fun part. [Volatility](https://code.google.com/p/volatility/) is an awesome tool for memory forensics. It’s free and written in Python, so it runs well on both Windows and Linux. On Windows you can either install Python 2.x to run Volatility or use standalone build which already contains required Python files and Volatility itself in one executable. On Linux you have to install Python (you should already have it). Volatility is already present in [Kali Linux](http://www.kali.org/)(BackTrack successor), for Gentoo there’s an unofficial ebuild available [here](https://github.com/dpzhang314/pentest/blob/master/app-forensics/volatility/volatility-2.2.ebuild). Place it in your local Portage overlay (f. e. /usr/local/portage/app-forensics/volatility/), create digest and install it:

**$** emerge -av app-forensics/volatility

Volatility is well documented ([Wiki](https://code.google.com/p/volatility/wiki/VolatilityIntroduction), [command reference](https://code.google.com/p/volatility/wiki/CommandReference22)), there’s also a quite nice [cheat sheet](https://code.google.com/p/volatility/downloads/detail?name=CheatSheet_v2.3.pdf) for it. All options and modules can be listed by using –help switch as well:

**$** volatility --help

Let’s take a sample memory dump from machine which got infected. At first let’s identify Windows version and system architecture:

**$** volatility -f /path/to/MEMORY.DMP imageinfo

The above command will return a number of strings, with one similar to this one:

Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)

Which means that machine is running Windows XP SP3 (installed with SP2 and updated to SP3) on x86 architecture. Now we can set the profile and location of image file:

**$** export VOLATILITY\_PROFILE=WinXPSP3x86

**$** export VOLATILITY\_LOCATION=file:///path/to/MEMORY.DMP

Thanks to last command we’ll not have to add the ‘-f /path/to/MEMORY.DMP’ every time. Let’s list all processes. We’ll be looking for processed which are hidden or their name is suspicious.

**$** volatility psxview

No hidden processes are on the list (there would be ‘False’ in pslist or psscan column), couple of them however looks interesting:

**Offset(P) Name PID pslist psscan thrdproc pspcdid csrss**

**---------- -------------------- ------ ------ ------ -------- ------- -----**

0x06b3c8c8 kyzu.exe               3284 True   True   False    True    False

0x06e42968 elazi.exe              5544 True   True   False    True    False

Smells like ZBot :-) This time we’re lucky, we know what we’re probably dealing with, but still we don’t have a sample and our AV doesn’t detect the threat. Let’s find parent process(es) for these two:

**$** volatility pstree

The most interesting part of results:

**Name Pid PPid Thds Hnds Time**

**---------------------------------- ------ ------ ------ ------ --------------------**

 0x86297020:explorer.exe             1384   3444     39    909 2013-05-13 12:51:55

. 0x86b3c8c8:kyzu.exe                3284   1384      0 ------ 2013-05-13 12:52:04

 0x86e42968:elazi.exe               5544   4696      0 ------ 2013-05-13 12:54:55

As you can see kyzu.exe was executed by explorer, but parent process for elazi.exe was already terminated and no traces are left in memory dump. Bad news are not over, all those ‘——‘ indicate that processes were already paged and we’ll not be able to simply dump them. Let’s check if psscan will find our missing parent for elazi.exe:

**$** volatility psscan

No luck – nothing here… No parent or other interesting process. Maybe we’ll have more luck with network connections?

**$** volatility connscan

Unfortunately not – all connections are either to local services or Microsoft servers.

There is an awesome plugin available in Volatility – it’s called malfind. It looks for injected code in processes within our dumped memory image. Let’s run it:

**$** volatility malfind -D /path/to/dump/dir

Command above will dump all the processes with injected code into a given directory. When it’s done, we’ll list all the files and sort them by size:

**$** cd /path/to/dump/dir

**$** ls -lS

What’s very interesting in here is that one sample with size 245760 Bytes was injected into multiple files, some of the are listed below:

-rw-r--r-- 1 h users 245760 05-28 23:59 process.0x86297020.0x11f0000.dmp

-rw-r--r-- 1 h users 245760 05-28 23:59 process.0x86b83b78.0xf30000.dmp

-rw-r--r-- 1 h users 245760 05-28 23:59 process.0x86b86af8.0xcf0000.dmp

-rw-r--r-- 1 h users 245760 05-28 23:59 process.0x86ba58a0.0xdf0000.dmp

Let’s upload one of these to [VirusTotal](https://www.virustotal.com/" \o "VirusTotal home page" \t "_blank) and… here it is – ZBot. Detection ratio: 21/47. Results can be found [here](https://www.virustotal.com/en/file/5333eb2df31b024319f8dcab02f8f6c6d19f78603900c3b90613a58a81221f58/analysis/1369778592/).

Now we confirmed that machine indeed is infected with ZBot. Let’s look at the process tree once again and search for memory offsets listed above (as a part of malfind dump names):

 0x86297020:explorer.exe              1384   3444     39    909 2013-05-13 12:51:55

. 0x86d79898:AcroRd32.exe 1348 1384 0 ------ 2013-05-13 21:25:29

.. 0x86b83b78:iexplore.exe 3668 1348 2 79 2013-05-13 12:52:04

. 0x86b86af8:ctfmon.exe               2260   1384      7    129 2013-05-13 12:52:04

. 0x86ba58a0:rundll32.exe             2708   1384      1     87 2013-05-13 12:52:04

After analyzing the tree of all processes in which the injected code was found, it’s easy to see which process is the root:

0x86297020:explorer.exe              1384   3444     39    909 2013-05-13 12:51:55

Unfortunately parent process of explorer.exe (with pid 3444) wasn’t found – it indicates that person that wrote this malware did the homework and finding the main process won’t be that easy.

Checking content of registry keys is also possible with Volatility. At first let’s see what are memory offsets of interested hives:

**$** volatility hivelist

Usually it’s good to focus on these:

0xe6562b60 0x55de5b60 \Device\HarddiskVolume1\Documents and Settings\User\ntuser.dat

0xe53e2b60 0x3f210b60 \Device\HarddiskVolume1\WINDOWS\system32\config\default

0xe18e46f0 0x0fd1f6f0 \Device\HarddiskVolume1\WINDOWS\system32\config\software

0xe18e36f0 0x0fcb66f0 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM

0xe53e6320 0x23486320 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY

0xe1036758 0x0b20a758 \Device\HarddiskVolume1\WINDOWS\system32\config\system

Depending on hive, you can f. e. look for apps added to one of many autorun keys (there dozens of lists with such keys, f. e. [here](http://www.bleepingcomputer.com/tutorials/windows-program-automatic-startup-locations/)). Let’s do that now and check one of the most popular locations:

**$** volatility printkey -o 0xe6562b60 -K 'Software\Microsoft\Windows\CurrentVersion\Run'

Now we’re really lucky – double hit!

REG\_SZ Symantec : (S) rundll32 "C:\Documents and Settings\User\Local Settings\Application Data\Mozilla\Symantec\dzhxvn.dll",startThreadW

REG\_SZ Ukywxua : (S) "C:\Documents and Settings\User\Application Data\Ocmu\kyzu.exe"

Second finding is for the kyzu.exe file, which we already know. Now we can fetch the file from infected machine and submit it for analysis on VirusTotal – it will most likely be the same as malware extracted by malfind.

The first finding however is something new. After quick digging we’ll find that dzhxvn.dll is a Trojan known as [AVKill](http://www.drwebhk.com/en/virus_techinfo/Trojan.AVKill.30237.html" \o "Dr.Web: Trojan.AVKill" \t "_blank).

Let’s see if we can get that dll from our dump:

**$** volatility dlllist | grep dzhxvn.dll

The force is still with us, Volatility found our dll:

0x02da0000 0x5e000 C:\Documents and Settings\User\Local Settings\Application Data\Mozilla\Symantec\dzhxvn.dll

0x02b00000 0x5e000 C:\Documents and Settings\User\Local Settings\Application Data\Mozilla\Symantec\dzhxvn.dll

0x02170000 0x5e000 C:\Documents and Settings\User\Local Settings\Application Data\Mozilla\Symantec\dzhxvn.dll

0x10000000 0x5e000 C:\Documents and Settings\User\Local Settings\Application Data\Mozilla\Symantec\dzhxvn.dll

Let’s dump it:

**$** volatility dlldump -r dzhxvn.dll -D /path/to/dump/dir

The dll was dumped multiple times, because it was allocated in memory more than once. For that dll VirusTotal shows detection score at 3/47 (results are [here](https://www.virustotal.com/en/file/ac8124d4692d42b6be0679415cc35387862921fc20ded8c5f01b10003fe3afa1/analysis/1369788678/)), from major AVs only Microsoft detects this file as Trojan:Win32/Tracur.AV.

There’s one more tool I’d like to mention here. It’s called strings and if you’re running Linux you have it in your system already, and in Windows you can find it in Sysinternals PsTools. It will examine any file for ASCII strings. Let’s run it on our ZBot sample:

**$** strings process.0x86297020.0x11f0000.dmp > /path/to/output.file

Now open the output file in your favorite text editor. In my ZBot sample I found this interesting string:

Coded by BRIAN KREBS for personal use only. I love my job & wife.

…and few others, which I’m going to cover in another article.

That’s it. I hope that you enjoyed reading my post, it covered only one (very basic) path for memory dump analysis, and for other malware it can look totally different – I’m going to present some other examples in the future.