



1 / 3 DO NOT WAST THE ENVIRONMENT

Forensic Challenge Writeup

Duckman

Contents

Section 1: Background	2
Section 2: Key Points.....	2
Section 3: Methodologies	2
Section 4: Results	2
4.1 File Identification:	2
4.2 Identification of Operating System Version.....	3
4.2.1 High Level Summary Generation	3
4.2.2 Identification of Operating Systems Version Information	3
4.3 Environmental Variables Extraction.....	4
 Figure 1: Hex representation of source file header	2
Figure 2: Snippet of output from CLI commands.....	3
Figure 3: Results from Volatilities imageinfo plugin	3
Figure 4: Output from kdbgscan plugin	4
Figure 5: Portion of output from Volatilities envvars plugin	4

Section 1: Background

For this challenge we are provided a source file to work with (5bd2510a83e82d271b7bf7fa4e0970d2.zip) and based upon the hint given to us within the title of the challenge (1/3 Do not waste the environment) we will be searching the binary for a specific environmental variable setting

Section 2: Key Points

- The source file is compressed using pkzip
- The uncompressed version of the file is a memory image taken from a Microsoft Windows system
- The challenge flag value is located within the environmental settings of several running binaries within the source file

Section 3: Methodologies

Initial validation of file contents was performed using the 010 Editor while the initial validation of uncompressed file contents was performed using both 010 Editor and the Linux strings command.

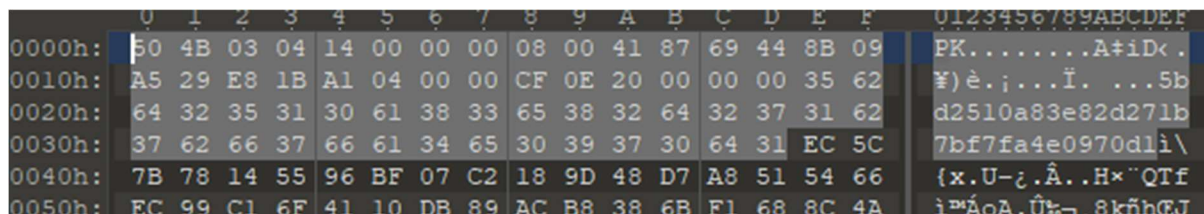
Final location identification and extraction of challenge flag was done using Volatility from within a Kali Linux VM.

Section 4: Results

Obtaining the desired challenge flag required a combination of a few different tools. This section describes which tools were used and how the tools output lead to the next step in the process

4.1 File Identification:

After downloading the source image from the challenge website, the target file was loaded into 010 Editor for examination of the file header for magic numbers. The results of this process (see Figure 1) quickly validated the fact that this source file's extension is a true representation of the file contents, a zip file



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	50	4B	03	04	14	00	00	00	08	00	41	87	69	44	8B	09	PK.....A+iD<.
0010h:	A5	29	E8	1B	A1	04	00	00	CF	0E	20	00	00	00	35	62	¥)è.¡...Ï. ...5b
0020h:	64	32	35	31	30	61	38	33	65	38	32	64	32	37	31	62	d2510a83e82d271b
0030h:	37	62	66	37	66	61	34	65	30	39	37	30	64	31	EC	5C	7bf7fa4e0970dlì\
0040h:	7B	78	14	55	96	BF	07	C2	18	9D	48	D7	A8	51	54	66	{x.U-¿.Â..H×"QTf
0050h:	EC	99	C1	6F	41	10	DB	89	AC	B8	38	6B	F1	68	8C	4A	ì"ÁoA.Ûh- 8kõhCFJ

Figure 1: Hex representation of source file header

Having confirmed the file contents of the original download the next step was to extract the final image from its compression container. For this, the file was copied into a Kali Linux VM where the command line utility **unzip** was used as the extraction program.

Finally, the following Command Line Instruction (CLI) was used to identify the Microsoft Windows Operating System (OS) as the source of the image (see Figure 2):

strings 5bd2510a83e82d271b7bf7fa4e0970d1 | grep -i -- 'Linux\|Windows' | more

```
root@kali:~/work# strings 5bd2510a83e82d271b7bf7fa4e0970d1 | grep -i -- 'Linux\|Windows' | more
SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings
SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\WinHttp
SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\WinHttp\Tracing
NtUserLoadImage{*.104C.8031.0.0.0.00000000.00008000};\ ; bit 15 PCI HACK DEFAULT_CARDBUS_WINDOWS
; subtractive decode, so its windows were ignored. This is no longer the case,
; and the BIOS configures the windows to claim enormous quantities of address
Microsoft.Windows.Diagnosis.Commands.GetDiagInput.Resources,1.0.0.0,en,31bf3856ad364e35,msil
Microsoft.Windows.Diagnosis.SDHost.Resources,1.0.0.0,en,31bf3856ad364e35,msil
Microsoft.Windows.Diagnosis.TroubleshootingPack.Resources,6.1.0.0,en,31bf3856ad364e35,msil
C:\Windows\system32\advapi32.dll[MofResourceName]
C:\Windows\system32\advapi32.dll[MofResourceName]
```

Figure 2: Snippet of output from CLI commands

4.2 Identification of Operating System Version

Now that the base OS has been identified (Microsoft Windows) the next step was to drill down and identify the exact (or as-close-as we can get it) versioning information.

4.2.1 High Level Summary Generation

Using the Volatility Memory Forensics Framework with it's *imageinfo* plugin command, whose output is shown in f3 below, the range of possible versions was narrowed down to four: Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1x86

```
root@kali:~/work# vol.py -a-f ./5bd2510a83e82d271b7bf7fa4e0970d1 imageinfo
Volatility Foundation Volatility Framework 2.6.1: Audit Policies from HKLM\SECURITY
INFO:poli: volatility.debug : Determining profile based on KDBG search...
Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1x86
AS Layer1 : IA32PagedMemory (Kernel\AS) from Real Mode memory
AS Layer2 : FileAddressSpace (/root/work/5bd2510a83e82d271b7bf7fa4e0970d1)
PAE type : No PAE system-wide notification routines
clipboardDTB : 0x185000L the contents of the windows clipboard
cmdlineKDBG : 0x82920be8L Process command-line arguments
Number of Processors : 1 Extract command history by scanning for COMMAND
1: Image Type (Service Pack) : 0
KPCR for CPU 0 : 0x82921c00L
Unit 24760 ( KUSER_SHARED_DATA : 0xffdf0000L
2: 38 (PATH) Image date and time : 2014-03-09 20:57:55 UTC+0000
Image local date and time : 2014-03-09 13:57:55 -0700
```

Figure 3: Results from Volatilities imageinfo plugin

4.2.2 Identification of Operating Systems Version Information

Having identified what OS the image file is from it was now time to identify the exact version information. For this, each of the suggestion profiles where used as input to Volatilities *kdbgscan* plugin (this plugin scans for the KDBGHeader signatures and uses the findings to eliminate potential false positives). As shown in Figure 4 below, with the contained KDGB signature within the source file Volatility was able to pinpoint the version to Win7SP1x86_23418

```

root@kali:~/work# vol.py -f ./5bd2510a83e82d271b7bf7fa4e0970d1 --profile=Win7SP1x86_23418 kdbgscan
Volatility Foundation Volatility Framework 2.6.1
*****
I Text Editor KDBG using: Kernel AS Win7SP1x86_23418 (6.1.7601 32bit)
Offset (P) bigpools : 0x82920be8 the keyboard buffer from Real Mode memory
KDBG owner tag checkcachedump : Trueumps cached domain hashes from memory
Profile suggestion (KDBGHeader): Win7SP1x86_23418ide notification routines
Version64 clipboard : 0x82920bc0 (Major: 15; Minor: 7600)ows clipboard
Service Pack (CmNtCSDVersion) : 0 Display process command-line arguments
Build string (NtBuildLab) : 7600.16385.x86fre.win7_rtm.09071anning for _COMMAND
PsActiveProcessHead : 0x82938658 (34 processes)
PsLoadedModuleList : 0x8293f570 (141 modules)
KernelBase (Hex - Ascii) : 0x82800000 (Matches MZ: True)
Major (OptionalHeader) : 6
Minor (OptionalHeader) : 1
KPCR 24763 (Hex - Ascii) : 0x82921c00 (CPU 0)
3: 214 (PATHEXT=)

```

Figure 4: Output from kdbgscan plugin

4.3 Environmental Variables Extraction

Using the results from sections 4.2.1 and 4.2.2 along with the Volatility plugin envvars (see fx below) the challenge flag was located and extracted.

```

root@kali:~/work# vol.py -f ./5bd2510a83e82d271b7bf7fa4e0970d1 --profile=Win7SP1x86_23418 envvars
Volatility Foundation Volatility Framework 2.6.1
File Type: data
-----
Pid Process Block Variable Information Value
-----
252 smss.exe atomp 0x003207f0 Path on and window station atomC:\Windows\System32
252 smss.exe atomp 0x003207f0 SystemDrive ion tables C:
252 smss.exe auditpol 0x003207f0 SystemRootudit Policies from HKC:\Windows
328 csrss.exe 0x002807f0 ComSpec C:\Windows\system32\cmd.exe
328 csrss.exe bigpools 0x002807f0 PP NO HOST CHECK using BigPageNO scanner
328 csrss.exe bioskbd 0x002807f0 NUMBER_OF_PROCESSORS from Real Mode memory
328 csrss.exe cachedump 0x002807f0 OS C:\Windows\system32\cmd.exe
328 csrss.exe callbacks 0x002807f0 Path e-wide notification routinC:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\
328 csrss.exe cmdline 0x002807f0 PATHEXT intents of the windows c.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
328 csrss.exe cmdscan 0x002807f0 PROCESSOR_ARCHITECTURE argumentx86
Zenmap(as root) xe 0x002807f0 PROCESSOR_IDENTIFIER scanning iX86 Family 6 Model 58 Stepping 9, GenuineIntel
328 csrss.exe 0x002807f0 PROCESSOR_LEVEL 6
328 csrss.exe 0x002807f0 PROCESSOR_REVISION 3a09

```

Figure 5: Portion of output from Volatilities envvars plugin