1 / 3 DO NOT WAST THE ENVIRONMENT

Forensic Challenge Writeup

Duckman

Contents

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

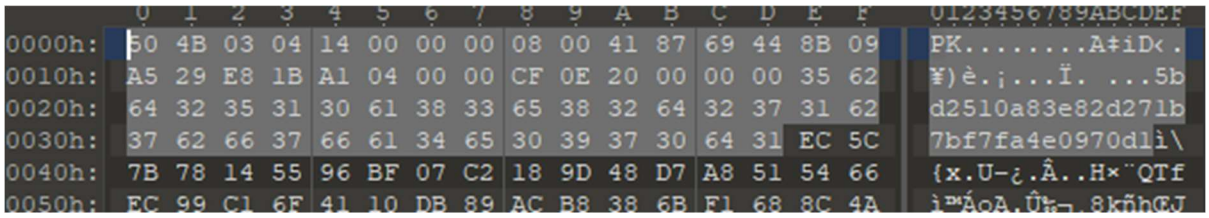

Figure 1: Hex representation of source file header

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

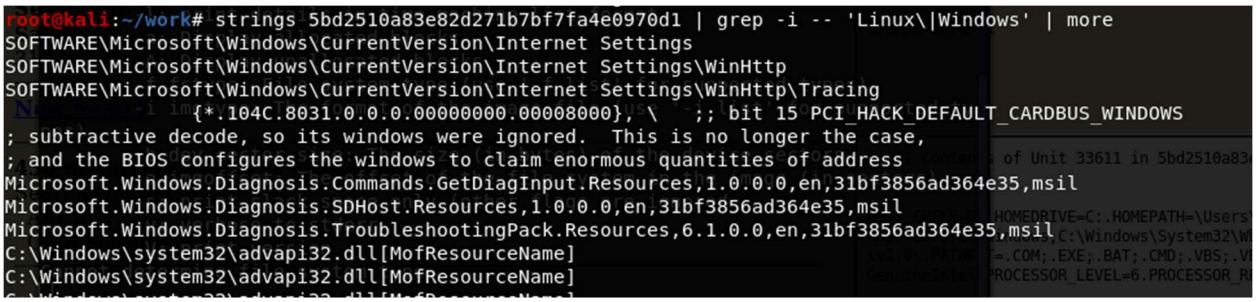

Figure 2: Snippet of output from CLI commands

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

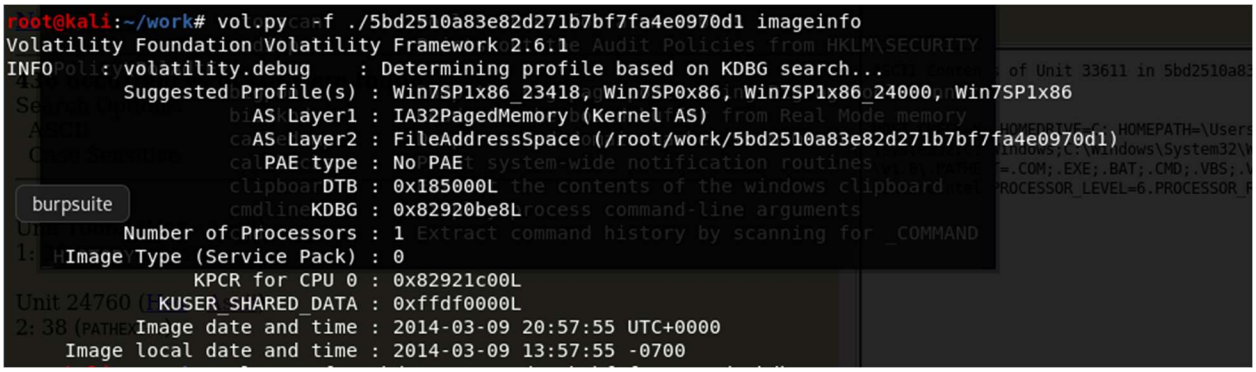

Figure 3: Results from Volatilities imageinfo plugin

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
**************************************************
Instantiating KDBG using: Kernel AS Win7SP1x86_23418 (6.1.7601 32bit)
Offset (V)                    : 0x82920be8
Offset (P)                    : 0x2920be8
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win7SP1x86_23418
Version64                     : 0x82920bc0 (Major: 15, Minor: 7600)
Service Pack (CmNtCSDVersion) : 0
Build string (NtBuildLab)     : 7600.16385.x86fre.win7_rtm.09071
PsActiveProcessHead           : 0x82938658 (34 processes)
PsLoadedModuleList            : 0x8293f570 (141 modules)
KernelBase                    : 0x82800000 (Matches MZ: True)
Major (OptionalHeader)        : 6
Minor (OptionalHeader)        : 1
KPCR                          : 0x82921c00 (CPU 0)
```

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 envars (see fx below) the challenge flag was located and extracted.

```
root@kali:~/work# vol.py -f ./5bd2510a83e82d271b7bf7fa4e0970d1 --profile=Win7SP1x86_23418 envars
Volatility Foundation Volatility Framework 2.6.1
Pid      Process            Block      Variable                       Value
-------- ------------------ ---------- ------------------------------ -----
     252 smss.exe           0x003207f0 Path                           C:\Windows\System32
     252 smss.exe           0x003207f0 SystemDrive                    C:
     252 smss.exe           0x003207f0 SystemRoot                     C:\Windows
     328 csrss.exe          0x002807f0 ComSpec                        C:\Windows\system32\cmd.exe
     328 csrss.exe          0x002807f0 FP_NO_HOST_CHECK               NO
     328 csrss.exe          0x002807f0 NUMBER_OF_PROCESSORS           1
     328 csrss.exe          0x002807f0 OS                             Windows_NT
     328 csrss.exe          0x002807f0 Path                           C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\
     328 csrss.exe          0x002807f0 PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
     328 csrss.exe          0x002807f0 PROCESSOR_ARCHITECTURE         x86
     328 csrss.exe          0x002807f0 PROCESSOR_IDENTIFIER           x86 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 envars plugin