George Giamouridis
gg1e20@soton.ac.uk
May 22, 2021

1 Task 1

1.1 Identifying the profile to be used to analyse the provided memory image

In order to identify the relevant profile to be used to analyse the provided memory image, we have to display some detailed information related to the memory image. To do so we will use the following command:

```
volatility -f memory.bin imageinfo
```

The -f parameter is used to specify the file (memory.bin) we want to analyse. We can now identify that the operating system from which this memory dump comes from is the **Win7SP1x64**, and therefore we can use that profile in our investigation. Now that we have identified the relevant profile we can use the following parameter to specify it and work with it:

```
--profile=Win7SP1x64
```

1.2 Identifying a malicious DLL

We can look for a malicious DLL by listing all the processes which were executed. The command we use in order to list all the processes is the following:

```
volatility -f memory.bin --profile=Win7SP1x64 pslist
```

If we examine all the processes we can spot an executable called **rundll32.exe** in the bottom of the list which seems suspicious because of its name. We now want to see what DLLs are invoked when this file is executed. We can list all the DLL's by using the **dlllist**. We use the following command to list all the DLLs:

```
volatility -f memory.bin --profile=Win7SP1x64 -p 1524 dlllist
```

By using **-p 1524** we define the process of which we want to examine. All DLL's are located inside the system32 folder apart from one:

$$dd4382d225a15dc09f92616131eff983.dll$$

We will move this DLL out of the vagrant environment in order to upload to VirusTotal and examine it. First we should dump the DLL by using **dlldump** and the following command:

```
volatility -f memory.bin --profile=Win7SP1x64
    --base=0x0000000180000000 dlldump --dump-dir ./
```

By using the –base we specify that we want to dump the DLL with base address **0x0000000180000000**. We now see that four dumps have been created. We only want the one with the **1524** PID. Now we can move the dumped DLL outside the vagrant environment by using the following command:

```
vagrant scp comp6236-lab3:~/module.1524.3f769610.180000000.dll .
```

We are now able to upload the DLL to VirusTotal and examine it. It is obvious that this DLL file is malicious as 41 out of 68 security vendors identified it as a malicious file.

1.3 PID of the process that's hosting the DLL

The PID of the process which hosts the malicious DLL can be easily found by looking at the first column of the results we found previously after running the following command:

```
volatility -f memory.bin --profile=Win7SP1x64 pslist
```

The PID of the process is **1524**.

1.4 Dumping the DLL and Calculating the SHA1 of the dumped file

In the previous question we dumped the DLL file and produced the **module.1524.3f769610.180000000.dll** file. In order to compute the SHA1 of this file we simply use the following command:

```
sha1sum module.1524.3f769610.180000000.dll
```

We have successfully calculated the SHA1 of the dumped dll which is

$$ab9fef8220f83d6df01e1d31019d53b563023be0$$

1.5 Analyzing the unpacked file

We are going to use the **strings** command in order to search for words that may be suspicious. We see in our results that there are some string related to some processes that. For example **CreateProcessAsUserW**. That means that the program somehow tries to start some processes. There are also some string related to request that are sent to the internet. For instance **HttpSendRequestA**. That means that the program tries to establish connections with something and send request.

1.6 Involvement with particular botnet

In order to check if this malware is related to a particular botnet, we will use the **strings** command. By using this command we will try to find suspicious information regarding this file. We will use the following command:

```
strings module.1524.3f769610.180000000.dll | less
```

We use the **less** parameter in order to displays the contents of one page at a time. After our investigation we see that there are two domains (mashevserv.com and ericpotic.com). We will try to find any online information about them in order to find any suspicious activity. We can clearly understand that this malware is related to a specific botnet called **Rovnix** (click to view the report) and the two domains we found are communicate with it.

2 Task 2

2.1 Identifying the profile to be used to analyse the provided memory image

Similarly to Task 1.1 we will use the following command in order to identify the relevant profile:

```
volatility -f dkom.mem imageinfo
```

We can now identify that the operating system from which this memory dump comes from is the **WinXPSP2x86**, and therefore we can use that profile in our investigation.

2.2 Discovering hidden processes

In order to discover any hidden processes we will use the **psxview**. The following command will list all the processes that are trying to hide:

```
volatility -f dkom.mem --profile=WinXPSP2x86 psxview
```

A process can be considered as *hidden* when the **pslist** and **psscan** columns have **False** value. In our case the process **"network_listene"** with PID **1696** is such a process, so it can be considered as the hidden process.

2.3 Process hiding techniques

The hiding technique which is used here is called **Direct kernel object manipulation (DKOM)**. This technique is a very common rootkit hiding technique for Windows operating systems. It actually hides third-party processes, drivers, files, and intermediate connections from the task manager and event scheduler.

By using this technique will not allow **pscan** to find the hidden process. We will run the pscan to see if this is true. We can see that even by running psscan, the DKOM technique is able to hide the malicious process and therefore is unable to be listed in the results.

2.4 Hidden processes and psscan

We can run the following command in order to see if the psscan can discover the hidden process:

```
volatility -f dkom.mem --profile=WinXPSP2x86 psscan
```

We can see that the pscan is unable to discover the hidden proccess **"network_listene"**.

2.5 Application compilation

We will first dump the hidden process with the following command:

```
volatility -f dkom.mem --profile=WinXPSP2x86 procdump
    offset=0x01a4bc20 --dump-dir .
```

Now that our process is dumped we can use the **strings** command

```
strings executable.1696.exe
```

and search for the user who compiled the application and the tool used. Inside all the strings that we get we can see that there is a path **c:/Documents and Settings/Brendan Dolan-Gavitt/My Documents/Visual Studio 2008/Projects/network_listener/Release/network_listener.pdb** which is related to the execution of the malicious process. Thus, we can see that the user who compiled the application is called **Brendan Dolan-Gavitt**.

We can also see that the tool which was used is the **Visual Studio 2008**.

2.6 Hidden process thread flow

We will use thr **strings** command again

```
strings executable.1696.exe
```

to find the flow of the thread. After we run the command we get a different string. However, we can notice that there are some strings revealing the flow of the thread. More specifically, the thread doing the following:

1. Establishes a TCP connection through a TCP socket with a remote machine (adversary machine)
2. Executes the test_compile.txt
3. Notifies that the testing has completed successfully
4. Sleeps for 5 seconds (delay)
5. Doing some calculations
6. Waiting for new user input