**Task 1 - Network Analysis**

Due to a malware attack, network analysis of the packet capture of the device from the Sales department was made using Wireshark. It is a tool for network traffic analysis, allowing for network troubleshooting, analysis, software and communications protocol analysis.

I will demonstrate how an infected packet can be found and detected using an example packet. The packet capture was downloaded from malware-traffic-analysis.net.

<https://www.malware-traffic-analysis.net/2022/03/21/2022-03-21-Brazil-sourced-malspam-infection.pcap.zip>.

1. **Wireshark**

Graphical user interface, text, application

Description automatically generated

Figure 1: Wireshark

Packet Capture files end with a .pcap extension. It is possible to open them with Wireshark; there, it is possible to view info about the different packets captured on the network, such as Time, Source Ip, Destination Ip, Protocol, length, and information about the packets.

1. **Column Preferences**

The format of the columns used by default is not the most efficient and does not display all the necessary info that we may need. By right-clicking on the header section and then selecting "Column Preferences…" we can edit headers to get the best information for our needs. Therefore, we will change the columns; the time will be displayed in UTC format, source port, and destination port.

Graphical user interface, text, application

Description automatically generated

Figure 2: Column Preferences

1. **Filtering packets**

On Wireshark, it is possible to take action on the packet data. One of the most powerful of these is filtering packets. The next step in our process is to filter the results. One of the most likely ways to get infected malware on systems through the network is by possibly clicking on a bad link or button, making HTTP requests, therefore, downloading undesirable content. Thus, we will filter through all the HTTP requests and responses in the search bar.

Text

Description automatically generated with low confidence

Figure 3: HTTP Requests and Responses

Looking through the filtered response, it is easy to see that a couple of the packets are simple and harmless, containing web error code responses. However, other packets have some unidentifiable info from a cursory glance.

Some of the packets will need a more in-depth investigation.

1. **Packet Info investigation**

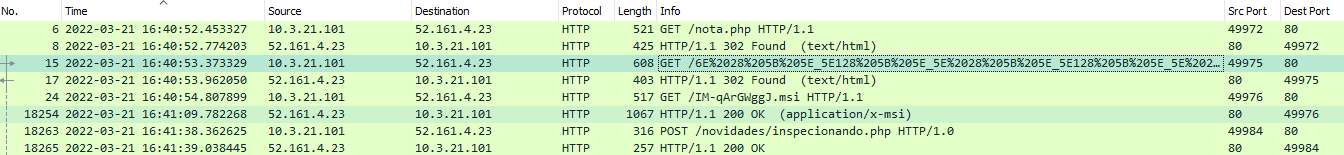


Figure 4: Packets with unclear info

As we can see from the filter results, there are eight results for HTTP requests. After a simple inspection of the Info header of the fifth packet, it can easily be seen that there is a file of some sort that ends with .msi. Microsoft Windows Installer(MSI) files can be used to install, maintain, and remove software applications. Therefore, it is concerning that a .msi file is contained in a packet coming to the system.

Another packet had a .php extension. Not n-knowing much about PHP files, it was best also to investigate this file.

1. **Extracting Packets**

Having discovered some suspicious packets with unknown results, the next best course of action would be to analyse the packets. To do this, we will first extract these two individual packets.

Graphical user interface, application

Description automatically generated

Figure 5: Exporting Objects

Export the objects into the assignments folder for easy access.

1. **Investigating objects**

With these two files that have been exported into the assignment folder, we now need to use a new tool to check if they are harmful or not.

Virustotal.com is an excellent online tool for scanning files and URLs and searching different IP domains or file hashes. The flexibility of input for the tool is first class. Virustotal uses many security software vendors to perform their analysis, including Avast, AVG, Microsoft, TrendMicro and many other leading security systems.

Graphical user interface, application, Teams

Description automatically generated

Figure 6: Virustotal Homepage.

First, we will analyse the .php file to check if its contents have anything that may be harmful. This can be done by simply uploading the file to the website.

Graphical user interface, text, application

Description automatically generated

Figure 7: PHP file

The result of the analysis informs us that there were no suspicious malware viruses or anything else of a malicious manner detected in the PHP file. This is a good result as we now know it is a safe file.

Next is the .msi file that was also found.

Graphical user interface

Description automatically generated with medium confidence

Figure 8: MSI Security Detection

Unfortunately, it seems that the MSI file was malicious. According to the analysis, it contained a trojan downloader, or spyware was in the installer, which is not something that is wanted on the system. This is a significant security risk.

The summary of the analysis also states that 27 security vendors and one sandbox have flagged the installer. The network traffic has been examined and detected for malware with this method.

Graphical user interface, text, application, email

Description automatically generated

Figure 9: MSI Security Summary

1. **Source Examination**

We must now examine the source of this malware that has been detected on the system of the Sales department. We can see the time this malware arrived on the system was at "2022-03-21 16:40", the source IP address "10.3.21.101", destination IP address "56.161.4.23", and the source and destination ports are "49976" and "80" respectively.



Figure 10: MSI HTTP.Request

To prevent this incident from recurring, blocking all traffic from this IP address should be considered.

**Task 2 - Memory Analysis**

**F1**

Get the volatility tool from volatilityfoundation.org and download it. Unzip the file into a location that can be easily accessed from cmd. Rename the executable file for easier usage in the cmd as well.

Download the Image and unzip the extracted file to the same directory of the volatility executable.

You are now ready to use the tool.

*Command: vol.exe -f ./"CA2 RAM Memory Capture for Task 2.raw" imageinfo*

This will determine the profile/machine type for the memory image.

Imageinfo offers a high-level summary of the sample for analysis. Through this, you can identify the operating system, service pack, and the system's hardware architecture, i.e. 32 or 64 bit.

**Text

Description automatically generated**

Figure 11

The profile of the Image can be found in the suggested profile section. For the initial analysis, we will use Win7SP1x86 as the image profile.

This tells us that the operating system was Windows 7, Special 32 bit version. This figure profile data will allow us to access further data about the Image.

**F2**

*Command : vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 pslist*

This will display the processes in the Image. It will retrieve critical data about each process, such as name, Process ID, Parent Process ID, Threads, Handles, Sessions, Wow64, and the start date and time of the process.

**Text

Description automatically generated**

Figure 12

Now we can review the result of this command and examineanything that is out of the ordinary. Some of the processes are related to the operating system and are necessary for the system's operation. Others are for applications such as the browser or background tasks such as chrome.exe and GoogleUpdate.exe.

Using *| find /c /v ""* at the end of a command can tell us how many lines are generated from the command. Therefore, using it, we found the number of processes that existed.

Forty-seven(47) processes were active.

Text

Description automatically generated

Figure 13

**F3**

***How many files were open at the time? How many jpeg files were available at the time?***

To discover the number of files that open at the time, we will use the filescan command. Filescan will find the open files even if hidden by a rootkit. The command's output shows the physical offset of the file\_object, file name, number of pointers to the object, number of handles to the object, and the effective permissions granted to the object.

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 filescan*

Text

Description automatically generated

Figure 14

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 filescan | find /c /v ""*

Once again, by counting the number of lines generated by the command, we can easily count the opened files in the memory image.

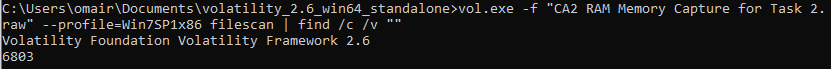


Figure 15

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 filescan | findstr "jpg"*

To find the files formatted in the .jpg format, I filtered the filescan result with the findstr command with jpg. This returns all the files which end with jpg.

Text

Description automatically generated

Figure 16

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 filescan | findstr "jpg"| find /c /v ""*

To count the number of files with the .jpg format, we add the *| find /c /v ""* filter again. There were only 25 jpg files.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 17

**F4**

***What searches appear in the Internet history? What browser was used in the search?***

Depending on the browser used, different commands will need to be used to get the browser history of the Image. Internet Explorer, Chrome, and Firefox have dedicated python scripts for searching browser history on volatility.

The scripts must be added to the plugins folder in the volatility directory.

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 chromehistory*

To search for the chrome history in the Image using the command above, we must first add a python script to the plugin folder of volatility. The chromehistory plugin can be found on GitHub.

<https://github.com/superponible/volatility-plugins/blob/master/chromehistory.py>

Text

Description automatically generated

Figure 18: Chromehistory

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 iehistory*

To get the browser history for internet explorer, enter the following command. The iehistory selector is built into volatility by default. This command will also bring other details such as time modified, accessed, and further information.

Text

Description automatically generated

Figure 19: Internet Explorer

Only Internet Explorer and chrome were on the system. Firefox and any other browser were not used on this system. This can be seen from the process list, as iexplorer.exe and chrome.exe could be found.

**F5**

***A file called duck.gif was downloaded. When did this happen? What program do you think was used to open the file? Give reasons for your answer?***

There are two files called duck.gif on the Image. The first appearance of duck.gif is in the pictures directory on "2013-01-23 17:27:11" this is the oldest version of the duck.gif file on the system.

Text

Description automatically generated

Figure 20: Pictures/duck.gif

The second appearance of duck.gif is in the downloads directory, which is on "2013-03-13 21:16:56". This is most likely the download referred to in the original question.

Text

Description automatically generated

Figure 21: Downloads/duck.gif

The two files were opened and accessed by three different programs: Internet Explorer, Chrome, and the Search Protocol. Internet Explorer and Chrome were most likely trying to find and access duck.gif; however, the Search Protocol is a Windows 7 process that is used to index files on the local drive making them easier to search.

Text

Description automatically generated

Figure 22: Search Prortocol

**F6 - *Choose a likely process and dump its memory to a file. Examine the dump with strings. Search for ASCII strings and UTF16 strings. Find something and point it out.***

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86* *procdump -p 2892 --dump-dir /Users/omair/Documents*

Using the procdump plugin we specify the proccess id which we will recoverfrom the memory capture. –dump-dir is a plugin that details whereto create the file, in our case we have creted the file in the same directory as volatility and the memory image. In this example, I will be dumping the chrome 2892 process.

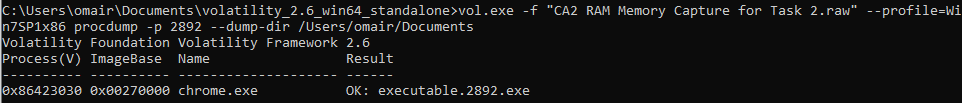


Figure 23: procdump

Before taking any other actions on the process dmp we must verify if is is safe and if it contains any malware. This is very important to always make sure that any data being handeled is always secure. Using Virustotal.com we scanned the process dump for any malware.

Table

Description automatically generated

Figure 24: Virus scan

*Command: vol.exe -f "CA2 RAM Memory Capture for Task 2.raw" --profile=Win7SP1x86 memdump -p 2892 --dump-dir /Users/omair/Documents/* *volatility\_2.6\_win64\_standalone*

The memdump above which is very similar to the procdump retrieves the memory dump of any given process. This dump will be saved in plain text which can be read.

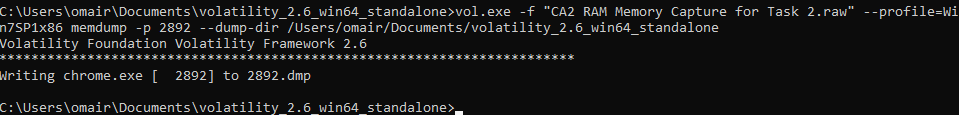


Figure 25: memdump

*Command: strings 2892.dmp |grep "duck.gif"*

The command above will help to filter the memory dump into what we want using grep. In this example I will be searching for duck.gif as we know that chrome was used to open the it.

*Text

Description automatically generated*

Figure 26: strings 2892.dmp | grep “duck.gif”