**MALWARE ANANLYSIS**

Static Analysis of the Emotet Malware

The md5 hash for this sample 92021ca10aed3046fc3be5ac1c2a094. You can confirm the file hash in REMnux without unzipping the file using the command “unzip -p emotet.doc.zip | md5sum” as shown below. The password for most zipped malware is “infected”.

Microsoft Office documents will either be in the legacy Object Linking and Embedding (OLE2) format or Open Office XML (OOXML) format beginning with Office 2007. Using the Linux “file” command you can see this is a Microsoft Word 2007+ file, which is OOXML.

OOXML files are actually ZIP archives containing the various file components. We can take a look at the file contents by unzipping the archive as shown below.If we look in the “word” folder, we see some interesting files and folders to include “vbaProject.bin” and an “ActiveX” folder. The presence of these indicate embedded macros and ActiveX.

We can use olevba or olevba3 to parse emotet.doc to find interesting information about its contents. Everything in this output is potentially concerning, but of note, we see that some code will run when the document is opened with autoopen() and it will attempt to hide itself from the user by using Windows Management Instrumentation (WMI) with ShowWindow. We also see that the code will create an object and potentially uses multiple methods of obfuscation.

We can use oledump.py to extract the macros from vbaProject.bin. We’ll start with zacGkX9 because it contained the autoopen() function. Running olevba.py with no arguments will display all of the VBA streams. Macro streams are marked with an uppercase “M”. Streams marked with a lowercase “m” only contain attribute or option statements. We can see below that zacGkX9 is stream 18. We’ll use oledump.py to extract stream 18 with the “-s <stream>” option and decompress it as readable text with the “-v” option. We can then append stream 17 with the same command using the append operator “>>”.Once extracted, we can use Visual Studio Code to take a look at streams 17 and 18. Below is the first few lines of the VBA code that will be run when the document is opened. You can see this is the stream zacGkX9 and the autoopen() subroutine is present

Emotet uses several code obfuscation techniques. The underscore character “\_” preceded with a space is used like word wrap in VBA. The malware authors use this technique in several places to obfuscate the structure of the code. This may seem insignificant, but malware developers will take advantage of nuance syntax to make the code as long and as complex as possible. The statement in figure 13 is used 10 times in the code

We can use Structured Storage Viewer to see what’s in those ActiveX objects. If we take a look at activeX7.bin, we see some interesting obfuscated code which PowerShell. If we load the extracted PowerShell into CyberChef, we can use the “From Base64” recipe to decode then use UTF-16.What the power shell does is, it first saves the download location on the victim computer to $filepath (originally $FwcAJs6). Then it creates a WebClient object with the handle of $webclient (originally $u8UAr3). It then saves 5 URLs to a list separated by @ symbols. Then it loops through that list to download the 284.exe file. It will stop looping through the list once it has successfully downloaded the file. If the downloaded file is over 23,931 bytes, it will launch the executable. 284.exe will begin the exploit process.

Emotet Memory dump analysis

./vol.py –f victim.raw imageinfo

This command will suggest suitable profiles for the file, in fact it check for the OS of the dump to parse it properly and shorten the list on arguments. So it’s clear that this is a dump of the windows OS. It has also given server OS as a guess but we will stick with no-server OS. So we got a profile as Win7SP0x64. Now we are ready to dig into this machine.

./vol.py –f victim.raw –profile= Win7SP0x64 psscan

So our first step will be to check all processes which were running on the machine, there are two arguments which we can pass to Volatility for this pslist & psscan. Difference between these two parameters is that pslist only shows active processes whereas psscan shows all terminated or hidden processes also. To reduce our time and make our search efficient we will use psscan. By looking at all processes I am only suspicious about one process, i.e. explorer.exe with PID 1860 since its PPID 1756 is nowhere in the list. I don’t know how valid this guess is but at this moment I have nothing else to be suspicious on. Next step should be looking at memory dump of our suspicious process but before this I would like to look at all active connection of this dump.

./vol.py –f victim.raw — profile=Win7SP0x64 netscan

By looking at this list of our suspicious process is increased as we can see PID 2464 is opening many unknown and malicious ports on the machine. There are another processes also 1004 but if you carefully look both processes share same PPID so I will just add one of them. One which has too many open connections i.e 2464.

./vol.py –f victim.raw — profile=Win7SP0x64 malfind

Malfind plugin of volatility detected the malicious processes or processes where malicious code may be injected based on multiple criteria VAD criteria. Like VADS tag and READ\_WRITE\_EXECUTE protection, these are strong indicators of the malicious processes