Static and Dynamic Malware Analysis

Submitted by
Kishan Patel
Undergraduate / Graduate
Information Technology / Computer Science
Gildart Haase School of Computer Sciences and Engineering

A **Capstone** paper submitted in partial fulfillment of the requirements for the course - **INFO 4205**

Under the supervision
of
Professor Jon K Morgan
Adjunct Professor, Department of Computer Science

FAIRLEIGH DICKINSON UNIVERSITY

(Spring 2024)

Abstract: I will perform static and dynamic malware analysis on a real malware specimen after learning how to use the different tools from the field of digital forensics.

General Description:

I will perform a static and dynamic malware analysis after learning from Tryhackme.com and its four modules, Basic Static Malware Analysis, Advanced Static Malware Analysis, Basic Dynamic Malware Analysis, and Advanced Dynamic Malware Analysis. Afterward, I will create a safe virtual environment with a VM hosting Windows 10 and Remnux and analyze the Zeus Banking Trojan while following a YouTube video. Different tools like PEstudio, Ghidra, Procmon, Procexp, and many more will be used to extract data and understand how the malware interacts with the machine and the different functions it uses to complete its task.

Background Information: Basic Static Analysis

On Tryhackme.com I completed the first module, Basic Static Malware Analysis. This module went over how to do a basic string search using PEstudio and the “floss -h” (FireEye Labs Obfuscated Strings Solver) command. Obfuscated strings are strings that are hidden to make malware analysis more difficult. It essentially hides the strings that are used to make a malware function. In malware, these obfuscated strings can have “greetz and shoutz” which can show the malware's real purpose, the author, and or affiliation. For example, a function name in the malware code can be “XYZcompany42ABC”, and in the middle would be the company name that the person is affiliated with. Additionally, a variable name can be a nickname that the malware author uses like a gamertag on a videogame. The “floss -h” command comes from Mandiant software which helps find the obfuscated strings using different deobfuscated techniques that would not be found using a string search. As practice I had to find how many obfuscated strings were found on the different samples provided by the module. Out of all of the strings found, I had to identify which string is the “DbgView.exe” executable.

Secondly, using PEstudio, the SHA256 hash value of a file can be found which is labeled as, “imphash”, which means “import hash”. This is the hash value for the specific file that is loaded onto PEstudio and using that hash value, you can go onto the website, “https://bazaar.abuse.ch/browse.php?search=imphash%3A756fdea446bc618b4804509775306c0d” which is a database of known malwares and see which malware family that file belongs to (if the hashes are the same). Next, fuzzy hashes are practiced using the “ssdeep.exe” command. The hash is calculated by breaking the file into small pieces and calculating the hashes of those pieces. When you have the ssdeep hashes, you can match these hashes together to find similar files. As practice, I had to find which imported hash (impash) was the same as another file in the malware directory provided by the module. Additionally, using the ssdeep tool, they have to see what the percentage match is for the files in the malware directory.

The next tool learned is Capa which helps identify the capabilities found in a PE file. Capa reads the file and tries to figure out what the behavior of the file is based on certain signatures like strings, imports, and other artifacts in the file. This uses different signatures found in files which can identify if the file is malicious based on its assumed behavior. When the Capa command is run, it displays the information it gathers according to the MITRE ATT&CK framework. The MITRE ATT&CK Framework is a globally accessible knowledge base of tactics and techniques based on real-world findings. It’s used across cybersecurity communities to create specific threat models and methodologies. The framework provides a common language for understanding and sharing information about cyber threat behaviors and helps to develop better cybersecurity defenses. One of the tables that it displays shows what the file can do like, delete files, get hostname, and parse the PE header. Again as practice, the module asked to see how many matches for anti-VM execution techniques were identified in the sample - if

the sample can suspend or resume a thread, and more.

Finally, the last section for this module does in-depth with PEstudio and PE headers. PE Headers are Portable Executables (PE files) that store programs in an executable file format. They are portable across systems with the same operating system (OS) and dependencies. The PE header, an initial part of the PE file, defines its characteristics and guides data reading. PE headers contain different linked libraries, imports, and functions that are used in that file. Many times PE files reuse code from libraries (often from Microsoft's Windows OS). These libraries are used in malware to get the malware to do certain tasks. These libraries control the functions of a computer and malware uses these libraries to complete its objective. This data is valuable for malware analysis. It additionally talks about packing which is a technique that malware authors use to reduce the effectiveness of some basic malware analysis techniques. Furthermore, combiners can group multiple items and they work closely with the packing technique.

Background Information: Advanced Static Analysis

In this module, I learned about Ghidra which is a free, open-source software reverse engineering tool that allows researchers to analyze compiled code to understand its functionality. It provides a way to decompile, disassemble, and debug binaries. Ghidra's features include decompilation into readable C code, disassembly into assembly language, a built-in debugger, and automatic identification of functions, variables, and other code structures. Additionally, Ghidra's layout has Program Trees, which show sections of the program, and a Symbol Tree, which contains important sections like Imports, Exports, and Functions. The imports section contains information about the libraries being imported by the program, while the Exports section contains the API/function calls being exported by the program. The Functions section

contains the functions found within the code, including the entry function, which takes the user to the start of the program they are analyzing. For practice, I had to upload one of the provided files and find the function call that displays a message in the message box, find what API call is in a certain .dll (dynamically linked libraries – a group of small programs that larger programs use to complete a task) under the imports section, and how many function calls are in the exports section as a way to practice the Ghidra software.

In the next section, I was asked to work with the assembly-level code. This section has me compare C and assembly codes, for example, printing “Hello World”, for loops, functions, and while loops. This helps review what some of the registers are needed for and how they work. When performing a static analysis, knowing what a loop, print statement, and function look like can be very useful. This section is simple because one of the courses that is needed in Fairleigh Dickinson University’s 4+1, IT and CS, program is assembly language. This course is focused on the basics of assembly, for example, variable declaration, for loops, while loops, function declaration, add, subtract, multiply, and divide. With this knowledge, the section has me practice how to find the virtual address of a function call, practice loops, and print statements using Ghidra.

The next section talks about the Windows API and what it does. The Windows API is a collection of functions and services provided by the Windows Operating System, which allows developers to create Windows applications. It includes functions like creating windows, menus, buttons, user-interface layouts, and as well as performing tasks such as file input/output and network communication. A common example of an API function is CreateProcess.
The CreateProcessA function is a part of the Create Process API that creates a new process and its primary thread. This function requires several parameters, including the name of the

executable file, command-line arguments, and security attributes. During a malware analysis, the identification of the API call and examination of the code can be crucial in understanding the malware's purpose. The section after this goes more in-depth about the different abilities of the Windows API and which malware uses what functions. For example, Downloaders are a type of malware designed to download other malware onto a victim's system. They can be disguised as software or files and spread in various ways. Downloaders use APIs like URLDownloadToFile (downloads a file from the internet), WinHttpOpen (requests to open a URL and returns an hinternet handle), WinHttpConnect (specifies the target HTTP server), and WinHttpOpenRequest (creates an HTTP request handle and stores its parameters in the handle) to download additional malicious code or updates to the malware. Keyloggers, for example, use several Windows APIs. Functions like SetWindowsHookEx (allows programs to monitor and react to system events), GetAsyncKeyState (checks the current state of a specific key, whether it's being pressed or not), GetKeyboardState (gets the status of all of the virtual keys), and GetKeyNameText (gets a string that represents a key name) allow malware to monitor and intercept system events, such as keystrokes or mouse clicks, and figure out the status or name of a pressed key. This allows the malware to capture sensitive information such as passwords and or credit card numbers. Finally, Data Exfiltration is unauthorized data transfer from an organization to an external destination, it is another common malware activity. Malware uses various Windows APIs, including InternetReadFile, which reads data from a handle that's opened by FtpPutFile, to steal data from a compromised system and transmit it to a different server. Understanding these APIs and their use can provide valuable insights during malware analysis.

Background Information: Basic Dynamic Analysis

The dynamic analysis module starts by explaining what tools are needed to perform the analysis. Dynamic malware analysis should be conducted in a controlled environment, ideally a virtual machine, to prevent accidental execution and intentional execution of malware. That is why it is necessary to use a sandboxed environment for analysis. Creating a sandbox requires an isolated machine, usually a virtual machine, that is disconnected from live or production systems and dedicated to malware analysis. The virtual machine should have the ability to save its initial clean state and revert to it once the malware analysis is complete, which is called a snapshot. This is necessary to prevent contamination of new malware analysis by infections from previous malware. The sandbox should also have monitoring tools to analyze the malware while it's executing inside the virtual machine. These tools can be automated or they can be manual and require an analyst to interact with them while performing the analysis. A file-sharing mechanism is also needed to introduce the malware into the virtual machine and send the analysis data or reports out. In this project, Oracle's VirtualBox is used with FlairVM which is running Windows 10, and Remnux, a forensic tool, both of which are on an isolated IP.

One of the applications that is used in basic dynamic analysis is ProcMon or Process Monitor. This tool is used to analyze the activity of a malware. It provides advanced functionalities for Windows and is widely used in security research. ProcMon is a simple tool to use because brief descriptions are available when the cursor hovers over a button. The tool provides options for opening and saving files that contain events, clearing all currently displayed events, and filtering the events shown in the ProcMon window. It also allows toggling on or off Registry, FileSystem, Network, Process/Thread, and Profiling events. ProcMon also allows the use of advanced filters. This is done by selecting filtering values, like Process Name, its relation,

value, and action. If the box is ticked, the filter is then applied, otherwise, the filter is ignored. This advanced filtering feature allows the results to be narrowed down to the events of interest, making the analysis process manageable and efficient.

The second to last section talks about API logger and API monitor. API logger provides basic information about the different API calls that are used by a process. API monitor does the same thing as API logger, however, it provides more advanced information about the process's calls. This section wants me to interact with the software and answer a few questions on how to find what API is being used for a particular process, how to find the path of the created file and more.

Process Explorer is the next tool that is discussed in the module. This tool allows the user to see their CPU utilization, memory usage, process IDs, Description, and the company name. When one of the processes is selected, using the process ID, the software displays a various amount of information about that process. It shows the handles the process has opened for different sections, processed, threads, files, mutexes, and semaphores. Additionally, this section talks about process masquerading which is when a malware author names a process similar to a Windows process so it becomes more difficult for an analyst to find, however, Process Explorer can make it slightly easier to find. Finally, this section talks about process hollowing which is a technique that malware uses that "hollows" an existing process, removes all of the original code from the process's memory, and replaces it with malicious code. So when an analyst looks at it, it looks like a real legitimate process, however, that process is running malicious code. Process Explorer is a great tool to help see if process hollowing is happening because it can check the strings that are present in the "image" and "memory", so if they are different process hollowing is happening, and if they are the same that is a legitimate process.

Finally, Regshot is a tool that helps find any changes that happen to the registry. It can check if registry keys were created, deleted, or modified during the dynamic analysis. The way it works is by first taking a snapshot of what the registry looks like before and after running the malware and then comparing the two snapshots. Whatever changes is what the malware did to the registry. A benefit of Regshot compared to the other tools used in dynamic analysis is that it does not need to be running while the malware is being executed, whereas some other tools have to be running. This is beneficial because some malware will check the processes on a system and if it notices an analysis tool running then the malware will not execute. Because Regshot is not running during the malware's execution, the malware will most likely do what it was intended. A negative of this however is that Regshot can only have the malware running in between the before and after snapshots. If another process is running in between those two snapshots then that can lead to a false positive.

Background Information: Advanced Dynamic Analysis

The advanced dynamic analysis module combines the information learned from the previous three modules and applies them, like changing the hash value, AV signatures, obfuscation of strings, runtime loading of DLLs, packing, identification of VMs, timing attacks, traces of user activity, and identification of analysis tools. Debuggers provide a malware analyst with the control desired to monitor a running program more closely, looking at the changes in different registers, variables, and memory regions as each instruction is executed which all relate to what is learned in the previous modules. This module provided me with a little bit of experience with FlareVM and one of its debuggers, x64dbg. It shows the program's assembly code with register values, a memory map, call stack, threads running, and any handles to files,

processes, or other resources. The sections in this module had me identify one of the evasion tactics used by one of the programs and then edit the assembly code so the evasion tactic does not work. This essentially makes the malware work even though it had an evasion tactic built in. Throughout this whole module, multiple questions are asked about the different tools described, and have me practice everything they have learned.

Hardware

The virtual machines will be running on a computer with the following parts:

- Motherboard: Asus TUF X-570 with WiFi
- Ram: Corsair DDR4 32Gb - Speed 2133 Megahertz
- CPU: Ryzen 9 5900X 12 Cores
- CPU Cooler: Deepcool AK620
- Graphics Card: MSI RTX 3060 12Gb GDDR6
- Hard Drives: Western Digital M.2 1 Tb and Samsung 2.5 inch SSD 1 Tb
- Power Supply: Corsair RM750 750 Watt
- Case: Corsair iCUE 220T Airflow - a total of 5 fans
- Operating System: Windows 10 Professional

Laboratory Set up

For this project a few software are used, Oracle Virtualbox VM, FlareVM, and Remnux. First, Oracle VM is a tool that can host virtual machines and that is where FlareVM and Remnux are going to be hosted. Second, FlareVM is a virtual machine developed by a cybersecurity company called FireEye and it comes pre-configured with a lot of different tools, software, and

scripts that are used often in malware analysis and reverse engineering. Third, Remnux is a Linux-based OS that has many tools for reverse engineering and a collection of tools that malware analysts can use. Remnux has many of the same tools that FlareVM has, however, some are unique to it. These tools in FlareVM and Remnux will be discussed later.

To set up the lab environment, I first downloaded the three software and then set up the Oracle VM. Once I had that setup, I used the Windows 10 ISO to create a new machine that has a second partition of 50 MB. After I was loaded into the Windows 10 machine I disabled the proxy auto-detect settings, Windows Defender in GPO, tamper protection, and the firewall. This is when I took my first snapshot of the machine so that if the FlareVM installation fails I can revert to this snapshot. Finally, to set up FlareVM, I downloaded the OS through Windows Powershell with the command: "(New-Object net.webclient).DownloadFile('https://raw.githubusercontent.com/mandiant/flarevm/main/install.ps1','$([Environment]::GetFolderPath("Desktop"))\\install.ps1")". Once it was downloaded I unblocked the file (Unblock-File .\\install.ps1), set the execution policy to unrestricted (Set-ExecutionPolicy Unrestricted), and then ran the install (.\\install.ps1). This process took a while, and when it was done I installed the default software that comes with FlareVM.

Next, I set up Remnux the same way I set up FlareVM except I did not create a second partition. Once Remnux was up and running I ran a few commands to create a fake network. I first typed, "inetism", to start the network service. I then edited the "inetsim.conf" file by uncommenting start_service DNS and setting the service_bind_address to 0.0.0.0. Next, I set the DNS default IP to the IP address of the Remnux VM 10.0.0.4.

Finally, I edited the virtual box network settings so that I could create an isolated host-only network adapter so that FlareVM and Remnux could talk to each other but not access

the internet. So in the Oracle VM virtual box, I clicked on tools, then network, and then clicked on create. This allowed me to create a new host-only network adapter. I scrolled down to the bottom of the virtual box and edited the IPv4 address for the host-only network adapter to 10.0.0.1. I set the DHCP (Dynamic Host Configuration Protocol, which automatically assigns an IP address to any devices connecting) to 10.0.0.2. The lower bound address was changed to 10.0.0.3 and the upper bound to 10.0.0.254. The last set was to go to the network setting for FlareVM and Remnux and change their network adapter to a host-only adapter. To test this I went to both virtual machines and tried to connect to the internet and was not able to do so which is good. I then pinged the Remnux machine with FlareVM and vice versa to make sure they both could communicate with each other.

Basic Static Analysis

Now that the lab is set up I was able to go to a github link and download the Zeus Banking Trojan. I downloaded it as a zip file and then extracted it to the desktop and its name was “invoice…pdf.exe”. I used the online tool, “Virustotal.com” to check if the malware sample, named "invoice," has been previously detected. The website also shows which security vendors have flagged the sample as malicious and provides some alternative names for the malware sample. Finally, the website provides the MD5, SHA-1, and SHA-256 hash of the file which can be cross-checked when you open the file up using PeStudio.

With PeStudio open, I selected the invoice file and compared the hash that it generates to the one that is on “virustotal.com”, and we can see that they are the same. Additionally, we can see that the file impersonates a PDF file when in reality it is an executable file. This was done on purpose because this is a banking trojan and someone working at a banking company would

most likely work with and open invoice files that are PDFs. Next, I checked if the malware binary is packed by comparing its raw size to its virtual size, and in this case, the malware binary is not packed since both of them are around 46000 bytes. I would know if packing/compression is taking place if the value in the raw size is smaller than the virtual size. Next, I moved over to the strings portion of PeStudio, which contains extracted plain text from the binary. There are some interesting function/API calls, such as "get capture" (meaning take a screenshot) and "write file," (write to an existing file) and other functions/API calls that are flagged, which indicate potential malicious functionality. Additionally, in the strings section of PeStudio, I found functions embedded within DLLs, for example, "create file," which is a function within the “KERNEL32.dll” that allows creating a new file on the file system. Above these DLLs, there are random “gibberish” characters before each DLL which might be the function names in the malware. These random function names could have been used to obfuscate what the program is doing. I then moved on to the libraries section in PeStudio which is based on the string section and found three different DLLs are being used, “SHLWAPI.dll, KERNEL32.dll, and USER32.dll”.

The next program I used was “cmder” and in this program, I used the “floss” command. I had everything that was found using the command outputted to a file and I then went through the file. This file has a lot of the same things that the strings section did in PeStudio, however, when used the “crtl+f” function and searched for any URLs I was able to find one called, “corect.com”. This is interesting since the word “correct” is not spelled correctly but it is being used as a URL. To check if the URL was malicious or not, I went to “virustotal.com” and searched for the URL. The results were surprising because “virustotal.com” said that no security

engines are flagging the website as malicious. I also used the Wayback machine or "web.archive.org" and searched the website, and again it resulted in nothing interesting.

Another program used is "capa" which was run in Windows PowerShell. What Capa is doing is that it tries to map the file and its contents to the Mitre ATT&CK framework which can tell the analyst what the file is doing. So when I ran the "invoice" file with Capa it came back saying that the ATT&CK Tactic that it is using is Defense Evasion and the technique is virtualization/sandbox evasion. This is so that malware analysts can not take this malware and try to figure out what it is trying to do. Additionally, the MBC (Malware Behavior Catalog) objective is anti-behavioral analysis and the MBC behavior is virtual machine detection. Finally, the capabilities it has is "reference anti-VM strings targeting VMWare" and "resolve function by parsing PE exports". All of that is used to make analyzing the malware more difficult as it is trying to actively avoid it.

Advanced Static Analysis

The next software that is used in the advanced static analysis is called "Cutter" (similar to Ghidra) which will show the assembly-level code for the specific file that is uploaded to the program. On the left-hand side, there is a column that will show the functions used, and under those functions that is where the assembly code is. To start, I started in the first function, called "entry0", and there I found an API call called "GetTickCount". This function most likely gets the time to see how long the computer has been on. Next, using the "gibberish" function call, "CellrotoCrudUntohighCols", and the "KERNEL32.CreateFile" dll, I tried to find it within the assembly level code. To do this, I went to the strings section and searched for "CellrotoCrudUntohighCols" and I found its location in the assembly-level code. Now to see if

“CellrotoCrudUntohighCols” and “KERNEL32.CreateFile” are related I tried to see if “KERNEL32.CreateFile” is near to “CellrotoCrudUntohighCols”. When I searched “KERNEL32.CreateFile”, I saw that it is only five lines underneath the “CellrotoCrudUntohighCols” function call. That means that they are related in some way. Using the list of all the “gibberish” function calls and the associated DLLs, we can figure out what the program is trying to do.

Dynamic Analysis

To start the dynamic analysis portion I first have to take a snapshot of the FlareVM virtual machine before the malware binary is executed, make sure I am not connected to the internet, and then run “inetsim” on the Remnux virtual machine so that it can simulate a DNS service running. The next thing I need to do is run “Procmon” which is going to show me all of the parent and child processes running on the machine. Now I can run the malware binary and after double clicking on the “invoice…pdf.exe” file I see that it wants to install Adobe Flash Player. Once it was done installing the “invoice…pdf.exe” file disappeared and I am assuming the binary deleted itself. Now that the program is running I go into “Procmon” and I find the “invoice…pdf.exe” process still running. It also has a few child processes, “InstallFlashPlayer.exe, WerFault.exe, cmd.exe, and Conhost.exe”. The “Conhost.exe” process is where a terminal session is open and it shows the command that was run, which is “\??\C:\Windows\system32\conhost.exe 0xffffffff - ForceV1”. The “InstallFlashPlayer.exe” process installs the flash player in the “\Temp\” folder and when I go into that folder that executable is still there.

Now that I have some evidence I can use a feature in “Procmon” which is its filter tool and filter all of the processes by their path and have it contain “C:\Users\Kishan\AppData\Local\Temp”. I notice everything that is being created, written, closed, etc. in that folder is done through the “invoice…pdf.exe” file and mostly for “InstallFlashPlayer.exe”, except one. I see that there is a DLL file that is written by the “invoice…pdf.exe” malicious file called “msimg32.dll”. I then go back into File Explorer find that DLL file and the “InstallFlashPlayer.exe” file and drag them to the desktop, and that is when I noticed they are greyed out meaning they are in a suspended mode. Finally, I go to “virustotal.com” and upload the “msimg32.dll”, fifty-one security engines flag this file as malicious. However, when I upload the InstallFlashPlayer.exe” file to “virustotal.com”, zero security vendors flag this file as malicious which makes me believe this file is legitimate.

I go back to “Procmon” and change the filter so that I am looking for operations containing “RegSetValue” for the “invoice…pdf.exe” process. This will find if the malware binary changed anything within the computer’s registry. The “invoice…pdf.exe” process sets keys for Google update and OneDrive processes. I assume that the malware is trying to establish persistence by running under these processes. I got to this conclusion because the parent process for Google update was, “wininit.exe” which happens on Windows start-up, so whenever Google update runs the “invoice…pdf.exe” file also runs.

Next, I move on to see if the malware binary tries to access the internet, and to do this I use the application “Wireshark” which can see all of the traffic on the selected network adapter. First, I launched “Wireshark” and then executed the “invoice…pdf.exe” file and monitored the traffic on the network. I first noticed two HTTP requests and the first one was a “get” request for the flash player that I saw during the static analysis part. While looking at this packet I see it has

a URL, “fpdownload.macromedia.com”, and I am assuming that is where the malware binary downloaded the flash player from. However, when uploading the URL to “Virustotal.com”, only one security engine says it is malicious. Using the Wayback Machine, “web.archive.org”, I was able to find nothing as well. Next, I used the command, “nslookup”, to get the non-authoritative server IP and put that into “virustotal.com”, and again, I got no result. So, when I put that URL into a web browser I go to a “helpx.adobe.com” website which makes me believe that is a real URL.

Zeus Banking Trojan History

The Zeus Banking Trojan was first created in 2007 when Eastern European hackers targeted the US Department of Transportation. It became public in 2011 and that is when the attacks took off. Since then there have been many variations of the Zeus Trojan and many other known families of the Trojan. The main purpose of the virus is to steal financial information by keylogging, then taking the infected machine and adding it to a botnet (a network of infected computers controlled by one individual through the internet), connecting to different servers to download more malware, etc. This malware variant caused around 11 million dollars worth of damage. (Kurt 3-5).

One of the versions of the Zeus Banking Trojan is called GameOver Zeus (GOZ). It is very similar to the one that I analyzed but it is slightly different. This malware spread through spam and phishing messages and was used by cybercriminals to get the same information as the original Zeus Trojan banking information like login information. When a system is compromised then it can send spam and or engage in distributed denial-of-service (DDoS) attacks. Older versions of the Zeus Trojan would use a botnet to work, but the GameOver Zeus variant uses

peer to peer (P2P) networks to infect hosts to communicate, distribute data, and uses encryption methods to evade detection. These peers work in a large proxy network to send binary updates, send configuration files, and send stolen data. Without a single point of failure, the GOZ's infrastructure makes taking it down more difficult. This is a only one other variant of the Zeus Trojan and combing all of the damage from all of the variants, the cost is in the hundreds of millions of dollars ("GameOver Zeus P2P" 2-3).

Future of Malware

In the past, malware has been created by humans, but since the rise of artificial intelligence (AI) malware is becoming more dangerous. Malware can be used to modifiy the action of the payload that the malware is going to deposit onto the system. Malware and AI can adapt to the conditions that it is deployed in to evoid dection and increase its effectiveness. Using AI capabilities improves the malware by making it more responsive to the environment its in. There are zero day and one day attacks which is when a vulnerability is recently discovered and AI can better it. AI can accelerate the discovery of zero-day vulnerabilties and the creation of the exploit. When these vulnerabilities are found, they are patched as soon as possible so cybercriminals want to launch these attacks as soon as possible, and AI reduced the time it takes to attack (Barry 11-12).

Works Cited

"Advanced Static Analysis." *TryHackMe*, 30 Aug. 2023, tryhackme.com/r/room/advancedstaticanalysis.

Baker, Kurt. "What Is Zeus Trojan Malware? - Crowdstrike." *Crowdstrike.Com*, 22 Jan. 2024, www.crowdstrike.com/cybersecurity-101/malware/trojan-zeus-malware/.

Barry, Christine. "5 Ways Cybercriminals Are Using AI: Malware Generation." *Journey Notes*, 17 Apr. 2024, blog.barracuda.com/2024/04/16/5-ways-cybercriminals-are-using-ai--malware-generation#:~:text=Using%20AI%20capabilities%20improves%20the,more%20responsive%20to%20the%20environment.&text=These%20are%20attacks%20against%20unknown%20or%20recently%20discovered%20vulnerabilities.

"Basic Dynamic Analysis." *TryHackMe*, 25 Apr. 2023, tryhackme.com/r/room/basicdynamicanalysis.

"Basic Static Analysis." *TryHackMe*, 14 Feb. 2023, tryhackme.com/r/room/staticanalysis1.

Collins, Grant. "Analyzing the Zeus Banking Trojan - Malware Analysis Project 101." *YouTube*, 16 Aug. 2023, youtu.be/USNOmFcebcU?si=NXtHSQif06Ex7b20.

Collins, Grant. "Build a Malware Analysis Lab (Self-Hosted & Cloud) - the Malware Analysis Project 101." *YouTube*, 9 Aug. 2023, youtu.be/rmSIm3BKu3Y?si=9HPaBIb7Ijje4F27.

"Download Virtualbox." *Oracle VM VirtualBox*, www.virtualbox.org/wiki/Downloads. Accessed 20 Apr. 2024.

"Dynamic Analysis: Debugging." *TryHackMe*, 23 May 2023, tryhackme.com/r/room/advanceddynamicanalysis.

"GameOver Zeus P2P Malware: CISA." *Cybersecurity and Infrastructure Security Agency CISA*, 30 Apr. 2024, www.cisa.gov/news-events/alerts/2014/06/02/gameover-zeus-p2p-malware.

Mandiant. "Mandiant/Flare-VM: A Collection of Software Installations Scripts for Windows Systems That Allows You to Easily Setup and Maintain a Reverse Engineering Environment on a VM." *GitHub*, github.com/mandiant/flare-vm. Accessed 20 Apr. 2024.

"Mitre ATT&CK®." *MITRE ATT&CK®*, attack.mitre.org/. Accessed 20 Apr. 2024.

"REMnux Documentation." *Get the Virtual Appliance*, docs.remnux.org/install-distro/get-virtual-appliance. Accessed 20 Apr. 2024.

Virustotal, www.virustotal.com/gui/home/upload. Accessed 20 Apr. 2024.

Wayback Machine, wayback-api.archive.org/. Accessed 20 Apr. 2024.