Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2022, 9(5):10-17**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Live memory Forensic, An Approach to Detecting Malicious Powershell

Damla Sunday Mador*[1], Okengwu U.A.[2], Mangai Sagai Bitrus[3]

*[1]National Open University of Nigeria
Madorict@Gmail.Com
[2]University of Port-Harcourt, Nigeria
Ugochi.okengwu@uniport.edu.ng
[3]University of Jos, Nigeria
Sagaimangai81@gmail.com

Abstract Currently, hundreds of thousands of new malicious files are created on a daily basis, existing pattern-based antivirus solutions failing to detect some classes of malware which leverage on PowerShell in other to execute fileless attacks. This is an issue of concern because PowerShell comes pre-installed on Windows machines, exposes strong functionality that attackers can exploit. To prevent these problems, live memory forensics methods of malware detection have been suggested in this work, this technique inspect the volatile memory (RAM) dump, created by Belkasoft Live RAM Capture, which are further investigated using volatility, an open-source memory analysis framework, written in python. It features a plugin system that enable us to develop plugins which work on user space memory, in other to extract forensic artefacts. The NIST digital forensics methodology, is used to implement this process through the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources. The memory forensics method has an advantage over the traditional antivirus program detection technique, because it has the able to discover sophisticated malwares which are stealthy enough to avoid leaving data on the computer's hard drive and obfuscate traditional antivirus programs.

Keywords Fileless Malware, Antivirus, PowerShell, Memory Forensic, Volatile Memory, Memory Dump

1. Introduction

Computers and other related computer systems (e.g. mobile devices, IoTs) that we use on a daily basis have become indisputable and essential components of our life, attracting the attention of cyber-attackers, which make use of malicious code. As a result, computer systems deployed in industries including military, health, entertainment, finance, and education have become targets for hostile actions such as unethical profit acquisition, information theft, and denial of service using various malwares.

One thing that has been consistent throughout the history of malicious codes, is the evolution of sophisticated malware program, which are fileless in nature. This type of malware differs from the traditional malware which reside in the computer Disk leaving footprint, that can easily be detected by the signature base method, this technique is quick and has the ability to detect known malware with a minimal false-positive rate (FPR). However, the signature-based technique fails to discover unknown malware with no signature, such as fileless malware and it is easily defeated by anti-analysis techniques such as evasion, encryption, and packing techniques.

The emergence of fileless malware has affected the threat landscape significantly. This malware has the ability to remain undetected in the system's main memory, making minimal changes to the file system. It takes

advantage of computer resources such as PowerShell, which is a command-line shell that supports a scripting language and is widely used for configuration management and task automation.

PowerShell is an object-oriented automation engine and scripting language. Since its release on 25th April, 2006, it has help IT professionals and system administrators to control & automate the administration of Windows Operating System and other applications. Administrative tasks in PowerShell are often executed using Command-lets (cmdlets), which are specialized .NET classes that implement a specific action. These operate by allowing PowerShell to access data from various data stores, such as the file system or registry, that have been made available to it via providers. PowerShell allows third-party developers to contribute cmdlets and providers. Scripts can utilize cmdlets, which can then be bundled as modules.

Due to the significant features of the PowerShell, Attackers are increasingly leveraging it to target computers, by "Living off the land" the concept of using legitimate tools like PowerShell readily accessible in almost all environments to achieve attacker aims.

PowerShell is also used by attackers because of its ability to run scripts remotely through WinRM, by executing payloads directly from memory making it stealthy and bypass endpoint security, leaving no trace after execution, thus anti-virus programs cannot detect it, due to anti-virus ability to only detect file-based malwares. Malicious script is also easy to develop, making it stress-free for attackers to deliver payload.

2. Related Works

Different research has been carried out on detecting malware, with some scholars focusing on File Base Malwares, which are malware that depends on executables to attack targeted computers.

Sung et al. [1] propose a method called Static Analysis for Vicious Executables (SAVE). The form of the signature for a given virus is given by a sequence of Windows API calls. A 32-bit number represents each API request. The most significant 16 bits correspond to the module, which the API call belongs to, whereas the least significant 16 bits corresponds to the API function's position in a vector of API functions. Sung et al. went further to compare SAVE to eight (8) different malware detection tools. Norton, McAfee Unix Scanner, McAfee, Dr. Web, Panda, Kasperksy, F-Secure, and Anti Ghostbusters were the malware detectors that SAVE was compared with. Sung et al. [1] put these scanners to the test against W32. Mydoom, W32. Bika, W32. Beagle, and W32. Blaster versions. Worm. SAVE was the only detector in the research that was able to detect all versions of the virus. In the work of Christodorescu et al. [2] malware signatures are represented by templates. Each template is made up of a three-part structure that includes instructions, variables, and symbolic constants. Templates try to generalize a malware instance signature and yet maintain the essence of the malicious code's behaviour. Christodorescu and Jha [2] proposed Static Analyser for Executables (SAFE) that has the ability to take patterns of malicious behaviour and turn them into an automaton. This automaton contains un-interpreted symbols in it that can later be bound to elements present in the executable being inspected. Honeycomb is a technique suggested by Kreibich and Crowcroft [3], which utilizes honeypots to build signatures and identify malware in network traffic. The authors' technique operates under the assumption that traffic which is directed to a honeypot is suspicious.

With the emergence of new categories of malware, which malware developers use PowerShell to accomplish their objectives, from reconnaissance via port scanning to privilege escalation by shell-code injection [4], persistence via registry editing, and payload delivery via a web client [5], According to a recent IBM research [6], over 57 percent of the attacks they examined were fileless, and many of these used PowerShell as an attack vector. This emphasizes the importance of detecting malicious PowerShell code.

The first detection of malicious PowerShell code was presented by Hendler, Kels, & Rubin, [7]. Their detector is based on a character-level representation DL model. which focus on identifying PowerShell commands, a detector of obfuscated PowerShell code was presented by Holmes and Bohannon, [8]. Although the challenge of identifying obfuscated scripts is similar to the problem of detecting malicious scripts, the two problems are distinct since many dangerous scripts are not obfuscated and hence cannot be identified using the technique.

Rusak, Al-Dujaili, & O'Reilly, [9] has proposed a malware classification system for malicious PowerShell scripts. Their classifier is based on a PowerShell script's Abstract Syntax Tree (AST) representation. A small-

scale embedding of 62 types of AST node types is used in their DL model. On their validation set, they claim an accuracy of 85%.

Rusak, Al-Dujaili, & O'Reilly, [9] used a DL model with an embedding layer to solve the challenge of identifying malicious Portable Executable (PE) files. Only the PE header is utilized in their classifier, and the raw bytes are fed into a DL model with a W2V-style embedding layer. Hendler, Kels, & Rubin (2019) introduced and tested a number of new deep learning-based detectors that use a pre-trained contextual embedding of tokens from the PowerShell "language." The embedding of these detectors is trained using a dataset enhanced by a huge corpus of unlabeled PowerShell scripts, which is a unique feature.

Krizhevsky et al. [10] and Sutskever et al. [11] make use of the AI-based malicious file detection involves two steps. Extracting feature data from the files is the first step. The second step is to use feature data to train the AI model for malicious file detection. There are two ways to extract feature data. The first method is to do a static analysis [12], while the second method is to conduct a dynamic study [13]. Stokes and Athiwaratkun [14] used dynamic analysis of PE files to solve the same problem by simulating file execution and documenting the sequence of system calls made by the application. This resulted in 114 distinct high-level system calls. These sequences were fed into a number of deep learning models, one of which contained an embedding layer that regarded each system call as a token.

For identifying malicious URLs, file paths, and registry settings, Saxe and Berlin [15] employ character-level embedding in conjunction with a Convolutional Neural Networks (CNN) architecture, which is a learning architecture, traditionally used in computer vision [16]. In order to simulate n-gram characteristics, they utilize a variety of filter sizes. Yang et al. [17] suggest a method for identifying malicious URLs that is DL-based. They employ a character-level embedding in their work, but they also explicitly examine 95 questionable tokens while fine-tuning it. Stokes et al. [18] propose a DL-based JavaScript and Visual Basic Script code detector. They utilize the script's byte format as model input. They tested two architectures: one that uses byte-level embedding, which is better at analysing relatively short code sequences, and another that analyses the input in fixed-length units before sending it to the embedding layer. The embedding was learned as part of the supervised training in both situations. A malicious JavaScript code detector is presented by Wang et al. (2016). Their detector transforms JavaScript code to binary vectors (based on ASCII character values), which are then fed into the DL architecture without using an embedding approach.

Choi [19] propose a new method for detecting malicious powershells using a Graph Convolution Network (GCN). He generated an adjacency matrix using Jaccard similarity between PowerShell scripts and provide a method to detect malicious PowerShell's using the adjacency matrix.

Pontiroli and Martinez [20] offer a way for detecting fileless malware by monitoring the system's behaviour. This system takes into account two factors. First, watch the security events for program execution via command-line console or PowerShell for processes with elevated privileges once they become live in memory.

This study addressed the detection of malicious PowerShell, using the Memory Forensics Technique. currently it is a popular technique and also prove to be efficient and accurate in malware analysis. It can examine malware hooks and code outside the function normal scope, Malware activities such as API hooking, DLL injection, and hidden processes can be detected using memory forensic techniques [21].

3. Methodology

Research methodology has many research dimensions and methods. In comparison to other forensic sciences, the field of computer forensics is relatively young. Unfortunately, many people do not understand what the term computer forensics means and what techniques are involved. In particular, there is a lack of clarity regarding the distinction between data extraction and data analysis. There is also confusion about how these two operations fit into the forensic process.

In this work we adopt the use of NIST (National Institute of Standard and Technology) digital forensics methodology. This methodology is scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources. The NIST methodology is divided into four steps, All the different stages are independently crucial and are linked together. The first step is the collection phase; this involves

the collection of evidence from the attacked computer. live memory forensic evidence is collected by creating the memory dump of the infected computer system.

The Examination step is the second phase of the forensic, this involve processing the working copy evidence that was collected and preserved utilizing various tools and techniques, following a defined, repeatable step by step process.

Analysis is the third step, in this phase, the processed evidence is analysed to answer the questions of the investigation of Who, What, When, Why, Where, and How. The forensics team analyses specific artefacts from the processed data depending on the type of investigation.

Now, that some information has been drawn based on the analysis of the evidence, the final phase is reporting, which is the process of preparing and presenting the information resulting from the analysis phase. In addition to fully documenting information related to the memory forensic, accurate record of all activity related to the investigation must be properly documented, including all methods used for testing system functionality and retrieving, copying, and storing data, as well as all actions taken to acquire, examine and assess evidence.

4. Output

a. Generating Payload

Undetectable PowerShell backdoor was generated with veil framework, veil has more than forth (40) different payloads that can be used, but for the purpose of our studies we use a PowerShell payload, *Powershell/meterpreter/rev_tcp. Py*.

The powershell payload was design to take advantage of window computer, through the powershell to executed malicious script. The payload uses reverse TCP to communicate with the target, it is also a meterpreter base payload which gives it the ability to run in the memory of the targeted computer leaving no or little footprint. Before the payload is generated it is important to set the ip address and port number of the attacker's computer in order to enable it listen and receive connection from the target as shown in figure 1. To complete the process of generating the payload, we need to input the name of the payload, which in our case we use "*freemoney*". The entire process of generating the payload was carried out on kali Linux OS installed on virtual Box machine.

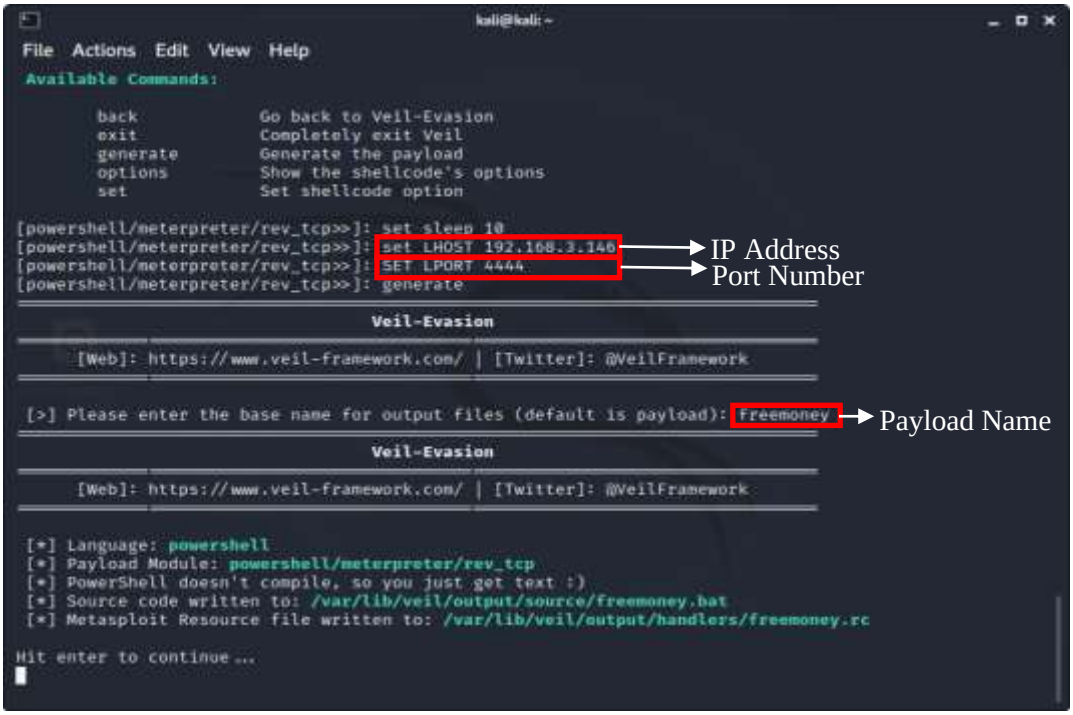

Figure 1: Generating powershell payload using veil Evasion

b. Payload Execution

When the payload is run on the target computer, it connects back to the attacker's computer through the IP address and the port number, of the attacker's machine. In figure 2, it shows the Metasploit handler is ready to receive any active session from the computer that runs the payload.

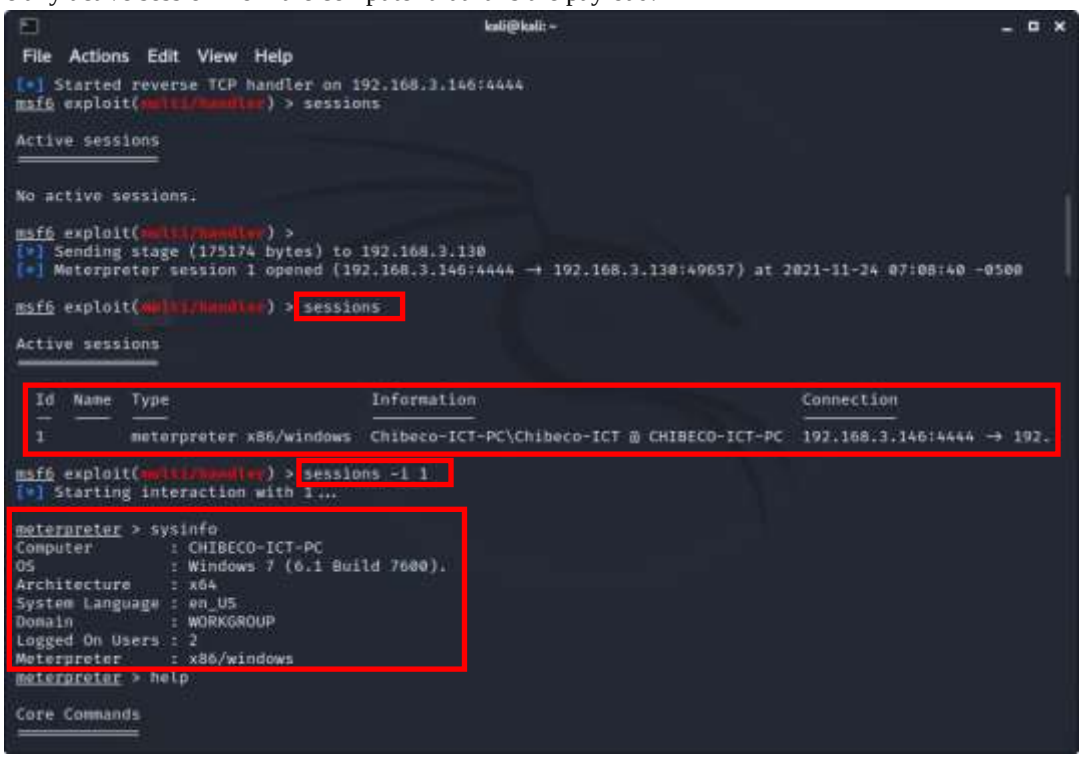

Figure 2: Metasploit handler connection to the target computer

In order to confirm any connection from the target computer, we run the command "*sessions*". This displays all the computer systems that executed our malicious PowerShell payload. We then select the particular computer we want to gain control of, from the list of the active sessions.

The command "*sysinfo*" is run to view information about the computer we have gained control of. The attacker has all the administrative privileges, just like the legitimate user of the computer system. He can add files, delete files, screen shot the computer screen, live stream the computer, take videos and pictures of the user, record the user's conversations, and lots more without the user or anti-virus programs detecting.

c. Creating memory dump

A memory dump of the compromised computer is created using dumpit, a memory dumping tool. The dump is further analysed in order to discover helpful information about the attack. It is important to create the dump on an external storage device and also make a duplicate so that the integrity of the dump can be preserved.

d. Analysing memory dump using Volatility

We used one of the most popular volatile memory software analysers, called Volatility, to analyse the memory dump created. This software assists us in retrieving useful information (such as the computer's running processes, legitimate processes, and various connections through which the infected computer communicates with). In order to get more information about the memory dump, we run ". */volatility –f memdump.mem imageinfo"* . "-f" specifies the dump file and "*imageinfo*" is a volatility plugin which displays information about the dump image such as the name of the computer, the operating system the computer is running, the time and date the dump was created, e.t.c.

Further investigation can now begin since we have the basis information about our dump. We can specify the OS profile (--profile=Win7sp1x64) using the command ". */volatility –f memdump.mem* –

profile=Win7sp1x64 pslist" we get all the list of processes that were running in the memory of the infected computer.

All the processes displayed seem to be legitimate, so we need to run another command to enable us to distinguish legitimate processes from suspicious processes. The plugin "pstree" helps us discover processes that are hidden in other processes or processes that depend on other processes to be initiated. Any process that begins with more than one dot (**.**) depends on the original process before it can be executed, and some malware takes advantage of this to hide their process. From figure 3, the process "*freemoney*" is suspicious because it has two different processes hidden, so we need to investigate that process further.

Figure 3: "ptree" plugin displaying processes and their parent processes

To also check the processes linked to each other, we use the "*joblink*" plugin. It prints process job link information. In figure 4, we discover that any time process **8148** is open, it executes processes **7784 and 4424,** so processes **7784** and **4424** are linked to process **8148**.

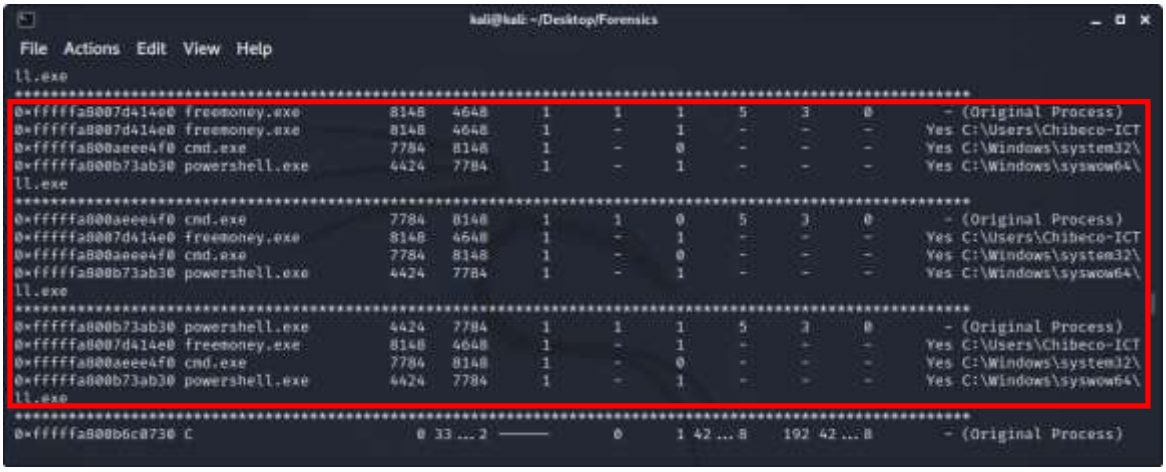

Figure 4: "joblink" plugin display

At this stage of the investigation, we will check the network connection that process **8148 (freemoney.exe)** is connected to. We will make use of the "*netscan*" plugin to scan the IP address and port number that the process is communicating to. As shown in figure 5, TCP connection was established with "*powershell.exe*", which is a process initiated by "*freemoney.exe*". The PowerShell is using port **4444** and is communicating with the destination IP address **192.168.3.130.**

So far, we have been able to detect how powershell.exe was launched by Process "freemoney.exe", which used a malicious script to run connection toward an external IP gaining control of our computer system, without antivirus programs knowing it.

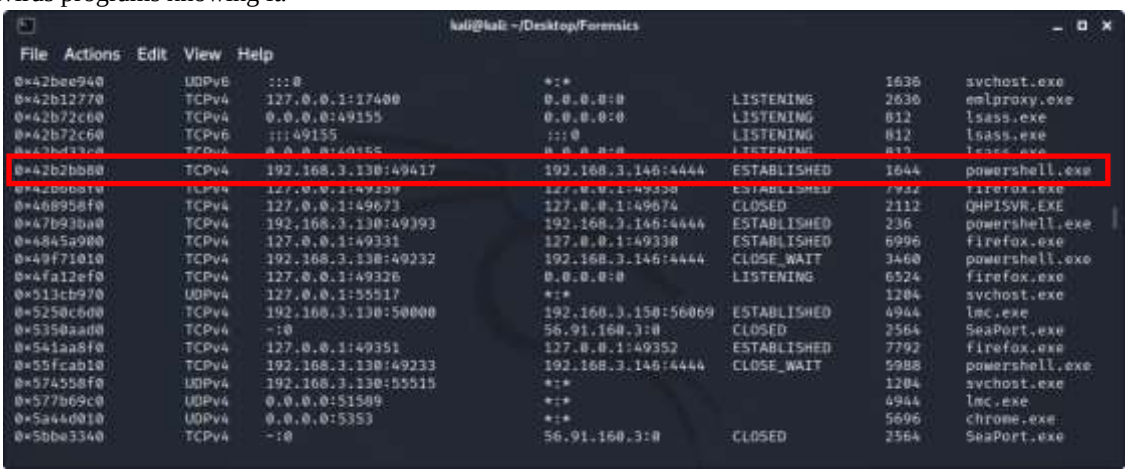

Figure 5: Network connection scan

5. Conclusion

Power shell comes in build with windows operating system. Therefore, our live memory forensics is limited to windows machine. In creating our memory dump its important to ensure that the computer system is powered ON, in other to capture the malicious PowerShell which is resident in the computer Main memory (RAM).

Memory forensics can be carried out using different tools and methods that differ based on the surrounding scenario, before we can start a memory forensics process, it is important we gather sufficient information about the suspected system which will guide us on the tools and the method to use for the forensics.

Examiner/forensics expert should use tools and techniques he/she is familiar with in accordance's to forensics best practice for effective results. Different artefacts may be discovered in the process of memory analysis that do not relate to the forensic, the examiner should put them in a separate dataset for further studies in order not to deviate from the investigation at hand.

As cyberattacks continue to advance and become more sophisticated, new technique used to detect and prevent such attacks are also fast emerging. live memory forensics is one of the methods that has proven to be effective, as different categories of malicious code are being detected including those that leave little or no footprint. This method has also been a great tool for incident response team to check the integrity of computing devices. I recommend that future work should be carried out on how to use this method to recover data being modify or deleted by cybercriminal, and also more studies can be done using Mac OS and other Operating systems in other to test the effectiveness of this technique across other platforms.

References

[1]. Sung, A., Xu, J., Chavez, P., & Mukkamala, S. (2004). Static analyzer of vicious executables (save). In Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC '04), 326–334.

[2]. Christodorescu, M., Jha, S., Seshia, S. A., Song, D., & Bryant, R. E. (2005). Semantics-Aware Malware Detection. in Proceedings of the 2005 IEEE Symposium on Secu- rity and Privacy (Oakland'05).

[3]. Kreibich, C., & Crowcroft, J. (2003). creating intrustion detection signatures using honeypots. In 2nd Workshop on Hot Topics in Network.

[4]. Palo Alto. (2017). Pulling Back the Curtains on Encoded Command PowerShell Attacks.

[5]. FireEye. (2018). Malicious PowerShell Detection via Machine Learning.

[6]. IBM. (2019). Ransomware Doesn't Pay in 2018 as Cybercriminals Turn to Cryptojacking for Profit.

[7]. Hendler, D., Kels, S., & Rubin, A. (2018). Detecting malicious powershell commands using deep neural networks. in Proceedings of the 2018 on Asia Conference on Computer and Communications Security. ACM, (pp. 187–197).

[8]. Holmes, D., & L., B. (2017). “Revoke-obfuscation: powershell obfuscation detection using science,”.
[9]. Rusak, G., Al-Dujaili, A., & O’Reilly, U.-M. (2018). Ast-based deep learning for detecting malicious powershell. in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. ACM, (pp. 2276–2278).
[10]. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Image net classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.
[11]. Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.
[12]. Gibert, D. (2016). Convolutional Neural Networks for Malware Classification. Master’s Thesis, Universitat de Barcelona. Barcelona, Spain.
[13]. Pascanu, R., Stokes, J., Sanossian, H., Marinescu, M., & Thomas. (2015). A. Malware classification with recurrent networks. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, Australia, (pp. 19–24).
[14]. Stokes J. W., R. A. (2018). Neural classification of malicious scripts: A study with javascript and vbscript. arXiv preprint arXiv:1805.05603.
[15]. Saxe, J., & Berlin, K. (2017). eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *arXiv preprint arXiv:1702.08568*.
[16]. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, vol. 1, no. 4, 541–551.
[17]. Yang, W., Zuo, W., & Cui, B. (2019). Detecting malicious urls via a keyword based convolutional gated-recurrent-unit neural network. IEEE Access.
[18]. Stokes, J. W., Agrawal, R., & McDonald, G. (2018). Neural classification of malicious scripts: A study with javascript and vbscript. arXiv preprint arXiv:1805.05603.
[19]. Choi, S. (2021). Malicious Powershell Detection Using Graph Convolution Network. MDPI, Basel, Switzerland.
[20]. Pentitol, S. M., & Martinez, F. R. (2015). The tao of .net and powershell malware analysis. In: Virus Bulletin Conference.
[21]. Khushali, V. (2020). A Review on Fileless Malware Analysis Techniques. International Journal of Engineering Research and Technology, 9.