POWERSHELL SECURITY REVIEW

KAMALESH RAM CHANDRAN GOVINDARAJ

INTRODUCTION:

PowerShell is Microsoft's new shell program which stands in as an update that replaces Microsoft's previous shell application, known as Windows Command Prompt (CMD). From the introduction of the Windows 7 operating system, Microsoft has been installing its machines with PowerShell. From its inception, it has proved to be a flexible system shell and a powerful scripting language. Thanks to its many cmdlets (command-lets), which are small sets of reusable scripts with unique functionalities, the system administrators are now able to automate administration and complicated tasks which was virtually impossible to do on a large scale using the CMD application. Since it is compiled using .NET, it makes it possible to access .NET assemblies and Dynamic Linked Libraries (DLL). Furthermore, PowerShell has many strong capabilities such as downloading content from remote locations, executing commands directly from memory, and accessing local registry keys and scheduled tasks.

Given all these reasons, it's no wonder why cybercriminals prefer to abuse PowerShell's prowess for their own illicit needs. One of the biggest issues with PowerShell based malicious attacks, happens to be the difficulty in logging the code executed by it. Although Microsoft has taken steps to improve the logging capabilities by introducing the AMSI (AntiMalware Scan Interface), we shall see later in this review that even the AMSI can be potentially bypassed, making the forensics analysis even harder. And this was just one of the many problems caused due to malicious actors misusing the strong functionalities of the PowerShell. One other common instance of abusing PowerShell's functionality is the code obfuscation of malicious commands. One of the papers being reviewed mentions the different ways in how a command can be obfuscated. All these pointers prove why we require an immediate need for ways to detect malicious code and to defer attackers. In this review we also take a look at some of the promising detection methods which use Deep learning approaches to detect malicious scripts written in PowerShell.

MICROSOFT AMSI:

AMSI stands for Antimalware Scan Interface. AMSI was introduced with Windows 10 and is implemented in Windows Defender. AMSI can be used to analyze dynamic scripting languages. AMSI is a generic standard interface that allows applications and services to interact with the antivirus solutions installed on the system. AMSI provides applications with the common techniques of an antivirus solution, such as scanning the hard drive and memory and analyzing content based on URL and IP address reputation checks. AMSI can also scan scripts that use tactics to conceal malicious code or layers of dynamic code.

POWERSHELL AND AMSI:

Introduction of the AMSI by Microsoft is designed to change the way we use Script-based languages such as PowerShell, VBScript or JScript in Microsoft Windows, which are usually easy prey for hackers. These scripting languages are integrated into the operating system, have a powerful range of functions and are also used to perform legitimate tasks. As a result, PowerShell has become a popular tool for hackers. When a user executes a script or initiates PowerShell, the AMSI.dll is injected into the process memory space. Prior to execution the following two API's are used by the antivirus to scan the buffer and strings for signs of malware.

1. AmsiScanBuffer()
2. AmsiScanString()

If a known signature is identified, execution doesn't initiate and a message appears that the script has been blocked by the antivirus software.

BYPASSING AMSI:

Microsoft implemented AMSI as a first defense to stop execution of multiple malware evasions that have been publicly disclosed. Since the scan is signature based, red teams and threat actors could evade AMSI by conducting various tactics. Even though some of the techniques in their original state are blocked, modification of strings and variables, encoding and obfuscation could bypass the filters set in place. As with any other security measures, there are ways to get around AMSI. A few methods that are discussed in this review are Powershell Downgrade, Base64 Encoding, Hooking, Memory Patching, Forcing Error and Registry key Modification

- PowerShell Downgrade

Even though Windows PowerShell 2.0 has been deprecated by Microsoft it hasn't been removed from the operating system. Older versions of PowerShell don't contain security controls such as AMSI protection and could be utilized as a form of evasion. Downgrading the PowerShell version to an older version is trivial and requires execution of the following command **powershell -version 2**

- Base64 Encoding

Fabian Mosch used an old AMSI bypass of Matt Graeber to prove that if base64 encoding is used on strings (AmsiUtils & amsiInitFailed) that trigger AMSI and decoded at runtime could be used as an evasion defeating the signatures of Microsoft. This technique prevents AMSI scanning capability for the current process by setting the "amsiInitFailed" flag.

- Hooking

Tom Carver created a proof of concept in the form of a DLL file which evades AMSI by hooking into the "AmsiScanBuffer" function. The "AmsiScanBuffer" will then be executed with

dummy parameters. The DLL needs to be injected into the PowerShell process which the AMSI bypass will perform.

- Memory Patching

Daniel Duggan released an AMSI bypass which patches the AmsiScanBuffer() function in order to always return **AMSI_RESULT_CLEAN** which indicates that no detection has been found.

- Forcing an Error

Forcing the AMSI initialization to fail (amsiInitFailed) will result that no scan will be initiated for the current process. Originally this was disclosed by **Matt Graeber** and Microsoft has developed a signature to prevent wider usage.

- Registry Key Modification

AMSI Providers are responsible for the scanning process by the antivirus product and are registered in a location in the registry. Removing the registry key of the AMSI provider will disable the ability of windows defender to perform AMSI inspection and evade the control. However, deleting a registry key is not considered a stealthy approach since it requires elevated rights.

MALICIOUS CODE DETECTION USING DEEP LEARNING:

Deep Learning is the extension of machine learning, only slightly more complex. Deep Learning models have three or more layers in their architecture which gives rise to the possibilities for a machine to learn some hidden characteristics in their input data, aiming to recreate how a human brain tries to understand the world and the objects around it.

Over the years, major strides have been made in the field of Deep Learning that facilitated many of the convenience features that have become commonplace today. One such application is Natural Language Processing (NLP), a part of the Artificial Intelligence (AI) field that focuses on making machines understand human speech and text in a colloquial sense. NLP is used for the purpose of analyzing sentiments by examining through online blogs, or any site that allows users to comment. In recent times, the combination of NLP along with Deep Learning techniques has provided some of the best results in accurately predicting the mood of the user. Since this method succeeds with online blogs, the authors of the paper, “Detecting Malicious PowerShell Commands using Deep Neural Networks”, feel like they can apply the same to detect malicious code in the PowerShell script, since we are dealing with strings of characters in both the cases and therefore by extension, should prove as a sufficient detection method.

The dataset chosen for the test consists of 66,388 unique PowerShell commands. Out of these, 6,290 commands have been labeled as malicious and the remaining 60,098 commands have been labeled benign. The authors examine various ML-based detectors, each which uses the various techniques of Deep Learning architectures. All seem to yield high success rates when it

comes to the detection performance of malicious commands, but some seem to slip by the detectors. This is especially true for detectors that use traditional NLP techniques such as linear classification, bag of words, etc. However, the authors claim that an ensemble learning detector that fuses traditional (NLP) and Deep Learning (CNN) methods, provides the best case detection results as it was able to identify the commands that slipped by the prior detectors.

The malicious dataset was split into two sets of 5,819 and 471 unique commands for the purposes of training & cross-validation and testing respectively. For the benign dataset the split was 48,094 and 12,004 for the same purpose. The encoded commands were decoded in the preprocessing stage before feeding the input into the model. The training model used a mini-batch gradient descent algorithm and the detection models used three different Deep Learning based detector architectures. One used a 9-layer CNN, one used a 4-layer CNN and the last detector used a Recurrent Neural Network (RNN) with some Long Short Term Memory (LSTM) blocks. While evaluating, the True Positive Rates (TPR) and the False Positive Rates (FPR) were recorded. After combining the traditional NLP based classifier (3 grams) and the Deep Learning based classifier (4-CNN), the authors were able to decrease the False Positive instances to only 7.

CONCLUSION:

Microsoft's introduction of the Antimalware Scan Interface closes a gap that until now has been exploited by use of script-based languages. AMSI allows PowerShell to be monitored and controlled in such a way that it is no longer lucrative for hackers. Microsoft has also included AMSI support in Office, so that macros can be scanned for known malware. AMSI provides a good way for application developers and makers of antivirus solutions to proactively prevent the spread of script-based malicious code in Windows. As for detection based on Deep Learning methods, the detectors gave optimistic results and proved they are more than sufficient for the job. The ensemble detector, which combines traditional and Deep Learning methods, was established as successful as it improved the already good results further by increasing the TPR and decreasing the FPR.

REFERENCES:

1. Danny Hendler, Shay Kels, Amir Rubin. Detecting Malicious PowerShell Commands using Deep Neural Networks. arXiv preprint arXiv:1804.04177, 2018.
2. https://www.blackhat.com/docs/us-16/materials/us-16-Mittal-AMSI-How-Windows-10-Plans-To-Stop-Script-Based-Attacks-And-How-Well-It-Does-It.pdf
3. https://docs.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps
4. https://0x00-0x00.github.io/research/2018/10/28/How-to-bypass-AMSI-and-Execute-ANY-malicious-powershell-code.html
5. Microsoft Corporation. Antimalware scan interface. https://msdn.microsoft.com/he-il/library/windows/desktop/dn889587(v=vs. 85).aspx, 2017.
6. Microsoft Corporation. Powershell. https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting?view= powershell-5.1, 2017.