Preventing File-less Attacks with Machine Learning Techniques

Alexandru - Gabriel Bucevschi
"Al.I. Cuza" University - Faculty of Computer Science
Bitdefender Cyber Threat Intelligence Lab
Iaşi, România
Email: abucevschi@bitdefender.com

Gheorghe Balan
"Al.I. Cuza" University - Faculty of Computer Science
Bitdefender Cyber Threat Intelligence Lab
Iaşi, România
Email: gbalan@bitdefender.com

Dumitru Bogdan Prelipcean
"Al.I. Cuza" University - Faculty of Computer Science
Bitdefender Cyber Threat Intelligence Lab
Iaşi, România
Email: bprelipcean@bitdefender.com

***Abstract*—The cyber-threat detection problem is a complex one due to the large diversity of attacks, increasing number of prevalent samples and to the arms race between attackers and security researchers.**

A new class of attacks which appeared in the past years is modifying its spreading and action methods in order to become non-persistent. Being non-persistent, the usual detection and analysis methods which are file oriented, do not work anymore. Therefore, several solutions became available like memory introspection, process activity monitoring or application enforcement. However, these solutions are time consuming, therefore their usage impose some additional resources needs.

In this paper we discuss an entry-level anomaly detection method of the command lines arguments which are passed to the most known system tools generally available in Windows and not only. Some of these tools are used for years in companies to automatize tasks, but only in the recent period they became a powerful tool for the attackers. The method is based on a derived version of Perceptron and consists in feature extraction and building a machine learning model.

***Keywords*-file-less attacks, perceptron, feature extraction, feature selection, malicious arguments, powershell, wmi, anomaly detection**

I. Introduction

File-less attacks are memory-level based attacks designed to avoid the classic and known detection solutions. The first found public reference dates to the summer of 2001 when Code Red was firstly discovered. From 2001 no major evidence was found until 2017 when the number of those attacks surged. Nearly 77% of successful breaches were based on file-less techniques and the success rate reached was almost 10 times higher than the already known file-based attacks.

The main property of this class is based on the fact the malicious code or payload is not stored at any moment as a file. A second property of this class is the use of benign, pre-installed, already available tools from the operating system. In this case no traces of malicious activity can be found before or after the attacks. This type of attacks uses system-tools like PowerShell, Windows Management Instrumentation (WMI) and other Windows administration tools and it achieves its persistence through modification of the registry hives and keys.

This paper presents a method to parse a command line, to extract relevant features and to build a machine learning model to detect the file-less attacks using a modified version of the perceptron algorithm.

II. Related work

Multiple researches and study cases where conducted regarding the file-less attacks. According to a report made by Ponemon[1], the majority of the attacks experienced by companies where file-less. The traditional methods of detecting malicious content were no longer effective, therefore new ways of detecting those attacks were needed.

Microsoft[2] came up with a solution based on combining standard techniques of detecting malware with next-gen techniques. These techniques include machine learning models on file behavior log, memory scanning with behavior monitoring and other dynamic defense systems included in Windows Defender APT. However, if we have to take into account the time needed to detonate the sample[1], the method could be time inefficient.

Another security vendor, Fireye, proposed a NLP pipeline to detect malicious PowerShells[3]. Their pipeline encapsulates a Decoder, a Named Entity Recognition, a Tokenizer, a Stemmer, a Vocabulary Vectorizer, a Supervised classifier (which contains different binary classification algorithms such as Kernel SVM, Gradient Boosted Trees and Deep

[1]detonating a sample is a way to force malware to execute in a Virtual Machine / Sandbox and make it behave like on a regular computer

Neural Networks) and, for the last layer, a Reasoning module in order to enable the analyst to validate predictions. However, the detailed structure of their learning networks is not provided, neither the accuracy and the specificity of their algorithms.

A more complex solution was proposed by Danny Hendler et. al[4] and it uses both NLP-based classifiers and Convolutional Neural Networks. They showed that the intricate obfuscation methods are hard to detect only by using NLP techniques, therefore the necessity of implementing a neural network. Even if the detection rate was high, we do not consider this method as a suitable one when taking into account a security product, due to its time and space complexity.

As we can observe, most of the presented solutions are based either on NLP, CNN or by analyzing the behavior of a software inside a controlled environment. Also, there is a lack of research regarding the presented topic mainly because of the freshness of the subject. Moreover, the main focus was on detecting malicious PowerShell commands. Therefore, the need to construct a model to detect complex file-less attacks tends to be neglected due to its complexity.

We will further present our solution which is based on parsing the command line in a linear time , extracting relevant features and building a model to describe a malicious command without focusing on a single evasion technique, neither on a specific language.

III. Problem description

The main element that makes file-less so popular is the easiness in which the launches initial infection and assists with the attacks using legitimate Windows administrative tools already installed on the victim‘s system. Typical attacks exploit vulnerabilities in browsers and associated programs. The user may become a victim of this type of attack when browsing an exploit-kit affected website that exploits vulnerabilities in browsers, Java, PDF reader, Flash player, script-based malware using JavaScript/JScript and other client-side software. Moreover, those attacks are successfully spreaded via spear-phishing emails. Spear-phishing is an email attack that targets a specific business, organization or individual pursuing unauthorized access to sensitive information. For example, Kovter is a file-less click fraud malware being, in fact, a downloader that evades detection by hiding in registry keys. Reports indicate that Kovter can have backdoor capabilities and uses hooks within certain APIs for persistence. Kovter was the most prevalent malware in the last two years, according to Center for Internet Security (CIS)[2],.

In the first trimester of 2019, it took second place in March, third place in January and April and seventh place in February.

[2]https://www.cisecurity.org

Being file-less, PowerShell attacks are powerful. Therefore an attacker can leverage this property to do significant damage to a system with a minimum effort. Besides Kovter, which is well known for the use of this technique, we have to mention ransomwares and even crypto-mining scripts which decided to use some of these techniques in the past months.

In their prediction for 2019, Watchguard[3] included a new variety of file-less malware which self-propagates with wormlike characteristics. They are called ”Vaporworms”, and so far, the ones detected into the wild propagates by installing copies of themselves on removable storage devices.

We previously presented how the malware industry around file-less techniques evolved. And we can see the potential and the exponential growth. However, PowerShell is not used only in malware industry. Being one of the most powerful scripting languages under Windows, it is also used in companies to automatize tasks, from the most important and complex to the insignificant ones. Therefore, our goal is to differentiate the clean scripts from the file-less attacks by detecting their anomalies. In order to achieve a small false positive rate, we will use a derived version of perceptron, One Side Class perceptron[5].

Consequently, we chose to use a modified version of the Perceptron algorithm named OSC, which incorporates an extra stage in the training algorithm to ensure that all the samples marked as benign are correctly classified. At each iteration, the OSC version of the Perceptron algorithm performs an extra training procedure for the records of a certain class, which is finished when the number of falsely classified entries from that class reaches a certain quota. In our training this minimization process is applied over the class of benign samples and the quota is set to zero. Furthermore, the evaluation of a sample in the OSC algorithm is identical to the one used in the Perceptron algorithm, which is an advantage from the standpoint of performance.

IV. Database and features

The main advancement of the current research consists in the manner of obtaining the data set of features. Much of the research has been the analysis of these tools because each instrument has a different command line semantics and each has to be interpreted differently to see what it is capable of. There are several tools that can download a file and some should not do so. For example certutil which is a tool that is part of the Certification Services, can use the urlcache argument to download a file.

We started with an initial dataset of 500,551 command lines, PowerShell scripts, Windows Management Instrumentation (WMI) scripts, Windows tasks, shortcut files (LNK),

[3]https://www.watchguard.com/wgrd-resource-center/2019-security-predictions

batch scripts and other files which contain command lines, provided by Bitdefender Cyber Threat Intelligence Lab and Virus Total Intelligence collected from February 2019 to May 2019.

Our dataset is remarkably unbalanced, since the number of clean commands is considerably higher than that of malicious commands. After applying permissive filters and removing inconsistencies[4] we remained with 499,550 command lines. The inconsistencies were generated by command lines with a relative small length. We decided to use 5:1 ratio for clean and infected files (for every malicious files we will have in our dataset 5 clean ones). This is highly important in order to reduce the rate of false positives. Using an unbalanced dataset and the One Side Class perceptron we are making sure that the chance of getting a false positive and affecting a clean application is reduced. According to our metrics 5:1 ratio reflects the ratio for clean and infected files which can be found in the wild. We have to mention that we conducted experiments with a smaller and higher ratio but we noticed that the choosen one fits best. This dataset was the one used in our trainings.

However, if we have to take into account the fact that the separation line between a file with anomalies and an anomaly-free one may be biased by duplicated sequences, we constructed a new dataset. This dataset was obtained by eliminating the duplicate sequences of features[5]. We remained with 33,451 command lines with unique features sequence.

We have to remark that the 5:1 ratio we used in building our dataset is a suitable one since we have to take into account that we have to reduce the number of false positive and avert over-fitting, which can result when a model is trained using a small dataset. Also, we have to mention that we tried to build datasets with a smaller ratio of clean/malicious commands (3:1, respectively 1:1) but this leads to a spike of false positives and we had to relinquish the idea and higher ratio but this leads to a diminution of detection rate without any false positives rate improvement.

To take into account our performance restriction we will extract only static features. We applied some preprocessing to get the command line cleaned of useless characters, we performed the concatenations after which we applied a lexical analysis in linear time because we iterate over command line only once. We extracted specialized features for malicious and clean command lines. We also extracted features that are common in both categories and that will help us to better observe the malicious commands, anomalies, obfuscation methods and to avoid false positive situations. We extracted a total of 340 features which are describing the command line structure(i.e. number of words, command line size), general behavior (i.e. process running the command line, parent process, used arguments) and specific traits(specific to malware attacks, i.e. downloading malicious payload, injecting system files, running crypto-mining in background). Some of the extracted features can be observed in Table I.

Description	Command line
Using encoded command	powershell -e WwBz...
Using alternating lower and upper case letters	pOWerSHell.eXE [sTrIng]::...
Start a miner	EthDcrMiner64.exe -epool...
Run a suspicious file	rundll32 ˜$ws.nfc,crys
Using obfuscation	powershell iex n‘e‘w-obj‘ect cmd.exe ˆ/ˆcˆˆsˆt ˆaˆrˆt ˆfiles.bat
Mshta suspicious javascript invocation	mshta javascript:R6TIiqZl...

Table I
EXAMPLES OF COMMAND LINES

We preferred to use only boolean features in order to increase the performance of the algorithm and also to reduce the memory used to save the data we needed. However, most of the features have integer values and therefore we have to discretize some of them in order to obtain only boolean features. The discretization process of a feature involves creating a set of intervals which enclose its values. The inclusion of a value within one of these intervals will create a new feature. The process used for discretizing the values of a feature aimed to create intervals which include elements whose frequency and value are very close. For example, taking into consideration the command line size we observed that the malicious command lines have either a small size to avoid detection either a long size to make complex operations. This empiric observation lead us to 17 intervals which results in 17 new features which will be used to represent the command line size in a boolean manner. For example, the first feature from the newly obtained ones will state if a command line is having fewer than 19 bytes while the last one will be extracted from those command lines which are having more than 10000 bytes. The entire distribution can be visualized in Table II.

Name	Interval	Feature Name
COMMAND LINE SIZE	$[1, 19)$	*"under-19-bytes"*
COMMAND LINE SIZE	$[19, 44)$	*"under-44-bytes"*
COMMAND LINE SIZE	$[44, 54)$	*"under-54-bytes"*
...	...	...
COMMAND LINE SIZE	$[10000, +\infty)$	*"higher-10000-bytes"*

Table II
EXAMPLE OF DISCRETIZATION

Therefore, from the total of 341 features we discretized a part of them and we resulted in a total of 534 features. By applying the Contextual Mutual Information Maximization criteria, we will select for our next step the most significant

[4]Inconsistencies are records with different labels and the same characteristics sequence

[5]Duplicates are different records with same label and features. Only one was kept

256 of them. This will improve time and space complexity in both training and testing phases.

By design, our feature extraction algorithm will extract at least 6 features for each sample. For example, for the *bitsadmin.exe /Tr^ans^f^er myD^own^lo^adJob "http://hidden-URL/m/2.png" "%cd%/54718.vbs"* command line we can see that the bitadmin process is used to download from a url which contains a file with the png extension but stores it in a file with the vbs extension. There is also a slight obfuscation of the command line using spaces and caret used for escape in the command line. We can extract at least this features: *"under-108-bytes"*, *"process-is-bitsadmin"*, *"url-extension-png"*, *"path-extension-vbs"*. These features are then feed into the training, in order to detect if the command line has an anomaly or not.

V. Results

As we mentioned before, the anomaly detection system was built around one side class perceptron. Its goal was to detect anomalies in command lines using only the extracted features and keeping a low rate of false positive. This is mainly due to the properties of the one side class perceptron which is aiming to exclude all the clean files with the risk of having some false negatives. In order to better document our research, we conducted five trainings from which we selected two to describe in this paper.

For the first one we used all the available sequences of extracted features. As we mentioned when we presented our dataset, we applied the Conditional Mutual Information Maximization criteria and we selected only 256 features from the entire collection of 534. For example, we decided to eliminate from training the feature describing the action of writing to a process memory (which is not seen as an anomaly) and to keep the one describing the usage of files with double extension (which is a major anomaly). In our training we used 428,787 sequences of features extracted from anomaly-free files and 71,274 sequences extracted from files with anomalies. We trained our model for 500 iterations and the obtained model developed, in the training phase, a sensitivity of 98.08% and an accuracy of 99.83%. The model will be named *Anomaly*.

For the second training we decided to keep only the unique sequences of extracted features, therefore no duplicates were allowed. We will name this model *UAnomaly*. From the initial collection of 428,787 anomaly-free files and 71,274 with anomalies we remained with 32,168 sequences of anomaly-free files 1,283 of files with anomalies. We again selected the most representative features using Conditional Mutual Information Maximization criteria and we trained this model for 500 iterations. The new model developed the best accuracy rate as it can be shown in Table III.

We tested our models on a new data feed provided by Bitdefender's technologies. From all the samples we selected 20,979 files labeled with anomalies and 16,567 labeled as anomaly-free within a week time period, from 01-May-2019 to 08-May-2019. We validated our models with our existent technologies and we obtained for the first model, *Anomaly*, a detection rate of 67.54% while for the second one, *UAnomaly*, a detection rate of 83.32% (Table IV).

Name	Sensitivity	Accuracy
$Anomaly$	98.08%	99.83%
$UAnomaly$	99.59%	99.94%

Table III
Comparative training results

Name	FP	FP rate	TP	Sensitivity
$Anomaly$	51	0.3%	14,169	67.54%
$UAnomaly$	54	0.3%	17,479	83.32%

Table IV
Results - Test Data Set

We can see that the model trained with the unique sequence of features has a better detection rate than the simple one. We can state the resulted model has a good behavior in a real world scenario, being a valid solution in the detection of file-less attacks using anomalies identification techniques.

We have to mention that three other trainings were conducted, but we had to discard them. In two of them we retrained the chosen models *Anomaly* and *UAnomaly* by increasing the number of iterations to 5000 but we obtained over-fitted models, while for the last one, we chose to map the features but no significant improvements were found when training for the same number of iterations.

VI. Conclusion

File-less attacks represent a genuine threat to traditional antivirus (AV) solution by using memory-level attacks without leaving any traces due to the fact that legitimate Windows Admin tool are being used in this process. By injecting legitimate Windows processes and altering system registry, file-less attacks can hide its presence and operate unnoticed.

In this paper we showed a solution based on a derived version of Perceptron in order to detect anomalies found in files which are containing PowerShell code. We can state that a file has an anomaly if it is not having features which can be observed in tasks used to automatize different process.

Taking into consideration that File-less attacks and zero-day vulnerabilities [6] exploited in these attacks are now considered the most dangerous threats especially targeted for the business side with increasing success ratio and causing considerable damage, the standard detection methods of blacklisting became inefficient. Looking at our results we can state that the at least *UAnomaly* solution is viable to be used in addition to the existing protection technologies

[6]A zero-day vulnerability is a computer-software vulnerability that is unknown to, unpatched by, or unaddressed by vendors.

as a complementary solution. Also, we can use *UAnomaly* model as an aggressive method with additional whitelisting techniques to detect and prevent File-less attacks.

Acknowledgments

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 830927.

References

[1] P. Institute, "The 2017 state of endpoint security risk," 2017.

[2] Microsoft, "Out of sight but not invisible: Defeating fileless malware with behavior monitoring, amsi, and next-gen av," 2018.

[3] Fireeye, "Malicious powershell detection via machine learning," 2018.

[4] D. Hendler, S. Kels, and A. Rubin, "Detecting malicious powershell commands using deep neural networks," *CoRR*, vol. abs/1804.04177, 2018.

[5] D. Gavrilut, R. Benchea, and C. Vatamanu, "Optimized zero false positives perceptron training for malware detection," in *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, 2012, pp. 247–253.