# IMShell-Dec: Pay More Attention to External Links in PowerShell

Ruidong Han, Chao Yang, Jianfeng Ma, Siqi Ma, Yunbo Wang, Feng Li

# IMShell-Dec:Pay More Attention to External Links in PowerShell

RuiDong Han [1(✉)], Chao Yang[1], JianFeng Ma[1], Siqi Ma[2], YunBo Wang[1], and Feng Li[1]

[1] Xidian University, Shannxi, China
{hanruidong,robertwang,fli1996}@stu.xidian.edu.cn
{chaoyang,jfma}@xidian.edu.cn
[2] CSIRO, Sydney, Australia
siqi.ma@csiro.au

**Abstract.** Windows proposes the PowerShell shell command line to substitute the traditional CMD. However, it is often utilized by the attacker to invade the victim because of its versatile functionality. In this paper, we investigate an attack combined PowerShell and image steganography. Compared with the traditional method, this attack can deceive the defender by hiding its malicious contents in benign images. To effectively detect this attack, we propose a framework IMShell-Dec, whose main target is to check external links before the execution of PowerShell script. IMShell-Dec trains a machine learning classifier with image examples, where the features are generated by merging histograms of three image color channels. Then IMShell-Dec examines the script through tracking and classifying the related images. The detector achieves more than 95% precision in 9,589 high-definition images.

**Keywords:** Intrusion detection · Powershell attack · Steganography detection

## 1 Introduction

Windows PowerShell is an adaptive and versatile command-line shell environment. It allows the user to take advantage of the .NET Framework [12,20], but it also provides additional functions for attackers to generate malicious scripts. Several open-source frameworks(e.g., *empire*[3], *nishang*[4], *PowerSploit*[5]) exploit it to attack victims. Traditional malicious scripts detection methods[5,1] rely on regular expression matching and complex rules. The regular expression is time-consuming to create while analyzing PowerShell script, and complex rules are hard to derive and pose a maintenance burden as the attack method evolves. Recently, several automated solutions have proposed to address these issues. Hendler *et al.* [8] leverages deep neural networks to detect obfuscated malicious

[3] https://github.com/EmpireProject/Empire
[4] https://github.com/samratashok/nishang
[5] https://github.com/PowerShellMafia/PowerSploit

PowerShell script. They encode characters as features to train a classifier. And Zhenyuan *et al.* [15] design a novel subtree-based de-obfuscation method to detect obfuscation, since the attacker always uses obfuscation to conceal their malicious contents. They implement obfuscation detection and emulation-based recovery in the abstract syntax tree. PowerDrive [19], a de-obfuscator for PowerShell attacks, recursively de-obfuscates the code by processing multi-stage de-obfuscation.

Previous works assume the payload exists in the form of script, however, we discover that attacker can mount their malicious PowerShell payload on a harmless medium outside of the script. Specifically, attackers may attempt to hide PowerShell malicious content in an external resource and use another harmless script to recover it later, which eliminates the distinctive characteristic caused by excessive obfuscation. In this work, we focus on the PowerShell attack combines with image steganography, where the attacker injects PowerShell script's information into the image's color channels, then generates another PowerShell release script to decode the malicious contents from the image. Both the release script and image itself are harmless, and, to improve stealthiness, the release script is usually embedded into a file (e.g., Office, JavaScript, C#) before delivered to the victims. When they run the file, the latent PowerShell release script retrieves the image and releases the malicious script. The malicious script can download Web files with the framework plugin *WebClient*, establishes remote control by sending requests to remote service, sets a persistence mechanism by creating a scheduled task or uninstalls a local application forcefully.

To counter this attack, we propose a novel machine-learning-based detection method, named IMShell-Dec. Unlike previous researches, which only consider the security of script itself, we also consider the external link, since the attacker can conceal their real malicious script in the external resource. We locate the external resource in the script, then apply a machine-learning-based method to check these external resources. We integrate the color histogram as the feature and train a classifier to identify malicious script.

The contribution is summarized in two folds. First, we research a new type of PowerShell attack. It hides the malicious script into an image and generates a standard release script, which can not be detected by the existing detection method. To address this emerging threat, we propose IMShell-Dec, which locates and identify the potentially malicious content hiding in the external image. IMShell-Dec achieves more than 95% precision in 9,589 high-definition images.

The rest of this paper is organized as follows. In Sect. 2, threat model of PowerShell attack including victim setting is introduced. Then, the detailed process of the threat is reported in Sect. 3. In Sect. 4, the detection mechanism is illustrated, which combines the image color histogram feature and machine learning. In Sect. 5, we describe the way we generate data samples, and report the detection performance of our method. Finally, relevant researches and conclusion are shown respectively in Sect. 6 and Sect. 7.

## 2 Threat Model and Scope

In this paper, we explore a novel attack combine PowerShell attack with image steganography. In this attack, the attacker generates two parts of the resource, including an image and a trap file with a release script. Then, the attacker spread the trap file through Web document, Webmail or USB device, and attempt to fool potential victims to give the execution permissions for the release script. The release script then decodes the malicious script from the image, which is hosted on a website or send to the victim along with the trap file. The whole attack flow is shown in Fig. 1.

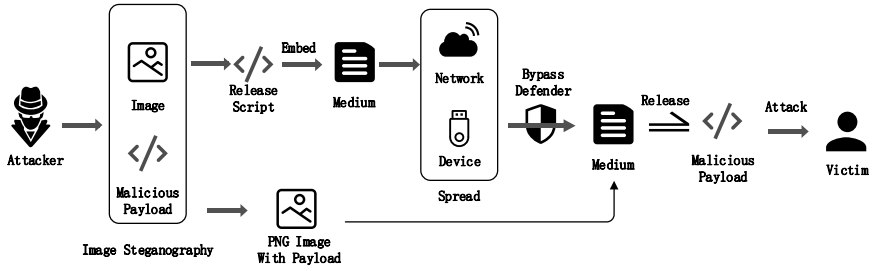



**Fig. 1.** System and threat model.

The scope of the attack is limited to the following scenarios. The target's system version is not older than Windows 7, since Microsoft developers set the PowerShell as a default application in the newer Windows version. The victim must be unaware or unfamiliar about the system security policy and proficiency of PowerShell. When victims get trap files, they accept to run it and granting necessary permission for the release script. For example, it is common for staff to download Office word documents from the Internet and open them with a local editor. When the document asks to allow update source or modify the file, the user often clicks sure button without paying attention to the prompts in the dialog box. Such action grants the file with specific permissions, allows the releasing script to retrieve a malicious payload and launch an attack.

## 3 Novel PowerShell Attack Through Image Steganography

In this section, we demonstrate the attack process through a concrete example and explain why the two parts of the attack can evade detection.

### 3.1 Principle of Attack

The conventional rule-based detection method mainly relies on the character form of PowerShell script to separate benign and malicious content. However,

image steganography allows the attacker to conceal their malicious payload in an external image, thus bypassing existed script detection. The attacker can then use a release script, which has no difference from the common benign scripts, to recover the payload and execute the intended attack.

In this work, we assume the attacker use *Invoke-PSImage*[6], a commonly used tool in the PowerShell, to generate the steganography image. *Invoke-PSImage* embeds the bytes of a PowerShell script into pixels of a PNG image by utilizing the least significant 4 bits of 2 color values in each pixel to hold the payload, then generates a release script that can extract the original payload later. If treated separately, both the release script and the image are harmless: the image is a PNG file, and the script's content is no more than a benign PowerShell command. The diverse format of the release script further strengthened the stealthiness, as the script itself can be a drop-in Office, VBScript, JavaScript, BAT Script, or a base64 certificate. Once the attacker lures the user to opening/running the file with certain permission, an image decoding command is executed in the memory without any GUI activity. The malicious payload is then extracted from the image existed in local or remote storage, and launch the intended attack.

As our threat model mentioned, the release script is embedded in another file to ensure it can sneak into the user system environment. For example, Windows provides several methods for data transferring between applications. One method is to use the dynamic data exchange protocol [10]. The DDE protocol carries out macro-less code execution in Office documents. Although Microsoft has limited it in ADV170021(2017.12) [7], there are still users who are not installing this patch. We conduct a pilot experiment on a colleague's computer, which is installed with Office 2013(15.0.4.4569.1504), and found out that the older version Office can run PowerShell code execution under the default permissions. *Excel4-DCOM* [8] enable raw shellcode execution on a remote Excel(32Bit), which opens the possibility to combines shellcode attack with lateral movement. JavaScript is capable of running PowerShell script by utilizing component "child process", it can also start a process to execute local PowerShell.exe to run a script. And .Net Framework also manages applications through SCM(Services Control Manager), where we can interfere PowerShell scripts in C# with public API.

### 3.2 Threat Usage

We perform experiments to determine the ability of the attack with three different forms of samples PowerShell scripts. At the same time, we explain why release scripts can slip away from the victim's attention and why image steganography makes the attack payload harder to be detected.

To verify the sensitivity of different defenders to scripts. We collect a corpus of PowerShell scripts (i.e., 4,079 PowerShell scripts in total) from

[6] https://github.com/peewpw/Invoke-PSImage
[7] https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/ADV170021
[8] https://github.com/outflanknl/Excel4-DCOM

*iocs*[9], which containing 27 kinds of malicious PowerShell scripts. The most frequently appeared script is *Downloader DFSP*, which downloads file with WebClient. To test the response of the defenders, we simulate a *Downloader DFSP* example as *iocs* provided, and process the example with different script forms, including an origin script, a base64 emending obfuscated script, and an image steganography script. In this simulation, we use this script to download the *7z*[10] application (and in the real attack, a malicious file) and execute it. More specifically, the origin script (see Fig. 2a) call WebClient to download the "7z.exe" into local directory "$HOME\Documents" and execute. The script is able to coding in Base64 (see Fig. 2b), which can directly be executed through PowerShell with the option "-enc". For the image steganography attack, we encode the script into an image's color channels through *Invoke-PSImage*, then generates a lossless PNG image and a release script (see Fig. 2c). Figure 4 compared the original image with its steganography processed copy.


(a) Downloader Script

(b) Coding Base64

(c) Steganography Image Release Script

**Fig. 2.** Example of scripts.

| Defender Name | Version | Origin | Base64 | Release |
|---|---|---|---|---|
| 360 | 12.0.0.2024 | ignore | warning | ignore |
| Kaspersky | 20.0.14.1085 | ignore | warning | ignore |
| Huorong | 5.0.28.1 | ignore | warning | ignore |
| Tencent | 2.0.6.27 | ignore | warning | ignore |
| Kingsoft | 8.29.18953 | ignore | ignore | ignore |
| MS Defender | 4.18.1907.4 | ignore | ignore | ignore |
| Norton | 5.16.1.3 | ignore | warning | ignore |
| McAfee | 4.0.127.1 | ignore | ignore | ignore |
| AVAST | 2.1.1286 | ignore | ignore | ignore |

**Fig. 3.** Reaction of defender.

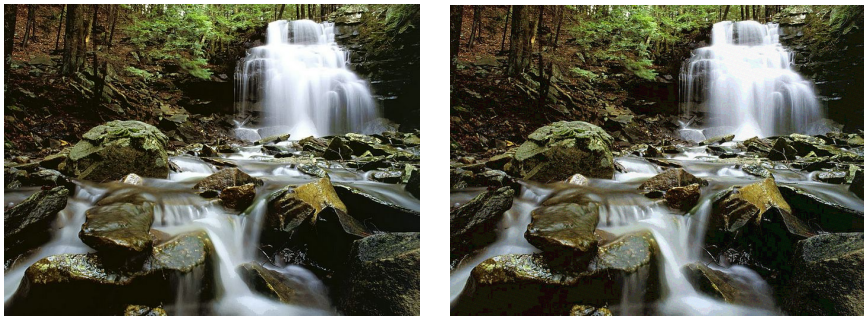
### 3.3 Effect of Attack

We evaluate the stealthiness of methods by observing the defender's response during the execution of scripts. Before this experiment, we download the latest defenders from their official websites and install them on Windows 10(1903) with PowerShell's version 5.1.18362. Nine experience results about security defender are enumerated in Fig. 3. We observe that all tested defenders do not raise a warning to the original script, a natural result since the script itself doesn't contain any abnormal behavior. However, defenders can easily intercept a naked malicious URL download attempt. Even we obfuscate the original

[9] https://github.com/pan-unit42/iocs/tree/master/psencmds
[10] https://www.7-zip.org/

(malicious) script with deep embedding, half of the defenders report that the script is operating suspiciously. This observation conforms with the discovery in research[8,11]. Image steganography conceals the true payload into a legitimate medium, extract it later through another independent and benign-looking release script, thus bypass the conventional script detection method.

As for the image, both defender and firewall only examine the script itself but pay no attention to its external image. Besides, as Fig. 4 shows, it is challenging to notice the blemish in steganographic image by naked eyes.



(a) Original Image (b) Steganography Image

**Fig. 4.** Comparison of original and steganographic image.

## 4 Our Proposed Defense Framework

To address the above PowerShell attacks, we proposed a machine-learning based defense framework, IMShell-Dec. In this section, we provide an overview of the proposed framework, and describe two key components of our framework: *feature extractor* and *detection model*.

### 4.1 Overview of IMShell-Dec

IMShell-Dec is a detection framework that aims to identify suspicious payload hiding in image. It starts by locating the external image links in PowerShell scripts. Once located, IMShell-Dec attempts to retrieve the image file, and determine whether there is a malicious payload in the image. The overview of IMShell-Dec is illustrated in Fig. 5.

When IMShell-Dec receives an unknown script, it starts by seeking for the external image links in PowerShell scripts and attempts to retrieve the image for any link located. Once the image has successfully retrieved, the feature extractor will transform images into useful features, and then the detection model will determine the category of these images. If the detection model label the

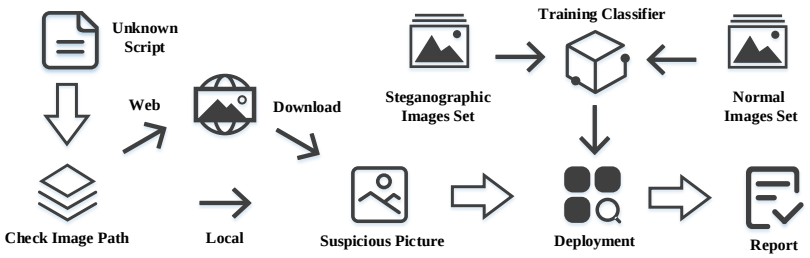


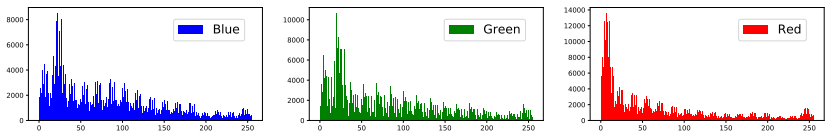

**Fig. 5.** Overview of IMShell-Dec

image as malicious, then IMShell-Dec will mark the source script as suspicious and raise a warning to the user. In the following subsections, we thoroughly describe the two key components of our proposed framework: *feature extractor*, and *detection model*.

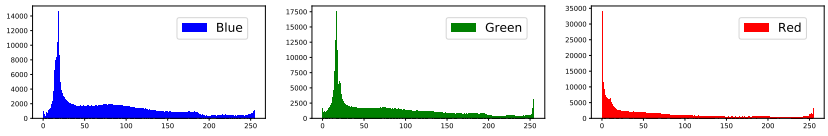
### 4.2 Feature Extractor

Before calling the detection model, we use feature extractor to distill useful information from the raw images. A pixel in the typical RGB-colored image consists of three integers, where each integer represents a colored channel with a range between 0 to 255. If we plot the number of pixels for each possible value, we obtain a frequency graph that represents the tonal distribution in a digital image. Such a graph is called "histogram".

Usually, the distribution in an unmodified image histogram tends to be smooth in general. However, steganography tools like *Invoke-PSImage* will introduce additional offsets to pixels, which may break the smooth shape of the distribution.



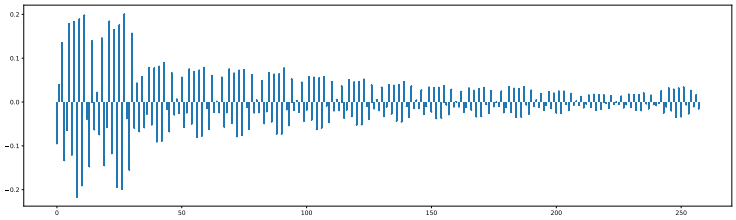

(a) Steganographic Image Histogram



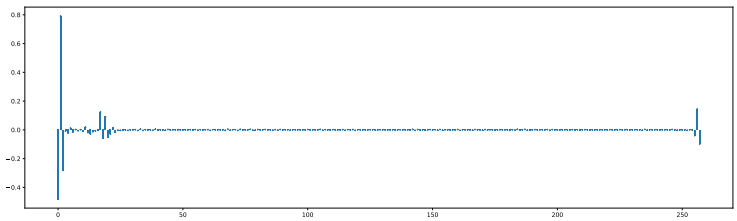

(b) Original Image Histogram

**Fig. 6.** Image color histogram.

To examine this conjecture, we record several image histograms and compare the smoothness of distribution before and after the steganographic process. As Fig. 6 shows, the steganographic process introduces numerous small yet obvious spikes in the image histogram. Hence, we leverage a filter with kernel $[-0.5, 1, -0.5]$ to process each color histograms, and transform the result of three channels into one feature vector, which reflected the smoothness of the transition between a particular value with its neighbor. To neutralize the influence of frequency scale, we further apply a min-max normalization and re-scale the feature vector to the range of $[-1, 1]$. The visualized image features are displayed in Fig. 7. It can be observed from the figure that the features extracted from a benign and malicious image are quite different.



(a) Steganographic Image Feature



(b) Original Image Feature

**Fig. 7.** Visualization Feature.

### 4.3 Detection Model

Once the image has processed into a feature, we can use a detection model to classify the images into two categories: benign or malicious. To obtain this model, we need to train it before the deployment with a training set.

A training set contains a set of images with a ground-truth label, where each image is processed forehand with feature extractor to generate corresponding feature vector. During the training process, these features are merged into one matrix, then the detection model takes the feature matrix as input and output the prediction. By comparing the prediction with ground-truth, the machine

learning algorithm is able to correct and update the detection model, thus improving its classification performance. When the training is completed, we freeze the model parameter and deploy the discriminative model to predict unseen images.

The label prediction process takes as input the discriminative model and a feature vector that the label is to be predicted. The discriminative model would assign the likelihoods of the feature to belong to each of the two categories. The category with the highest likelihood would be outputted as the predicted label for the feature. The prediction result of the feature is the judgment of the image.

There are several machine learning algorithms able to perform the classification task. In this study, we select three algorithms, namely Linear Discriminant Analysis (LDA), Random Forest (RF), and Back-Propagation Networks (BPNs) as our experiment candidates. Their detection performance and time consumption will be evaluated in the next section.

## 5 Experiment of IMShell-Dec

### 5.1 Experiment Setting and Metrics

We implement our code in Python 3.7 and perform the experiment on a PC equipped with Intel i7-9700 CPU. We use *iocs*, a corpus contains 4079 PowerShell scripts as our malicious script database, which has an average size of 312 Bytes and can be divided into 27 categories by the attack behavior. Then, we collect 5,510 high-definition images from the Internet, and randomly select 4,079 of them to generate synthetic copies with malicious payloads. This image dataset(9,589 samples in total) is used for classifier training.

To ensure the fidelity and reproducibility of the experiment, we apply k-fold cross-validation to generate a diversified dataset for both training and evaluation. Specifically, the entire dataset is randomly split into $k(k = 10$ in our work) subsets, each subset contains approximately 958 images, with roughly 550 benign and 408 malicious. We then generate ten distinctive data groups by taking each unique subset as test data while the remaining subsets as training data. We train and evaluate models on each data groups, and report the average performance.

We use accuracy, precision, recall, and F1-score as the performance metrics, which are defined as follows:

$$Accuracy = \frac{TP + TN}{P + N} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

where $P$ is the number of malicious image, $N$ is the number of benign image, $TP$ is the number of correctly identified malicious image, $FP$ is the number of benign images that incorrectly labeled as malicious, and $FN$ is the number of malicious image that wrongly classified as benign.

We also apply the receiver operating characteristic curve (ROC), area under curve (AUC) and precision/recall curve (PR) to assess the overall effectiveness of IMShell-Dec. ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied, and AUC, the size of the area under the ROC curve, represented the model's capability to distinguish between classes. Both indicators of the PR curve focus on positive examples in a binary classifier system. If the PR of one classifier is entirely covered another, it can be asserted that the classifier has better performance than another. In past studies, F1, accuracy, recall and precision scores of 0.8 or above are often considered reasonable (e.g., [13,3,17]).

### 5.2 Result

We implement and evaluate three classification algorithm, including linear discriminant analysis, back-propagation network, and random forest. Experiment results are shown in Table 1.

**Table 1.** Metrics of each classifiers.

| Classifier | Accuracy | Precision | Recall | F1 | AUC |
|---|---|---|---|---|---|
| LDA | 0.918 | 0.898 | 0.943 | 0.920 | 0.972 |
| RF | 0.941 | 0.927 | 0.959 | 0.943 | 0.988 |
| BPNs | 0.961 | 0.954 | 0.968 | 0.961 | 0.993 |

**Table 2.** Average training time.

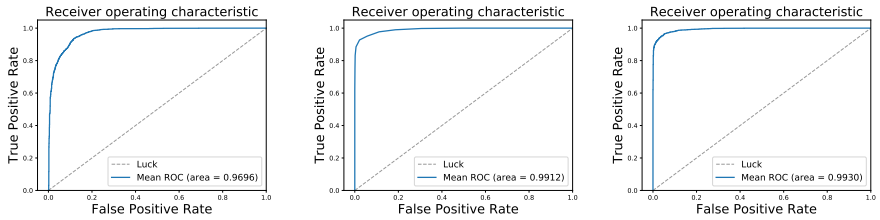
| Classifier | Training (S) | Predicting (S) |
|---|---|---|
| LDA | 0.31077 | 0.00151 |
| RF | 0.72302 | 0.01240 |
| BPNs | 5.67275 | 0.00747 |

We note that the results of these classifiers are reasonably good, which means our proposed scheme performs well in classifying images. Among the three machine learning algorithms, their F1 results achieve 0.920, 0.943, and 0.961 (out of 1) respectively. The result of LDA is lower than the other two algorithms. Through the ROC, AUC and PR results shown in Fig. 8 and Fig. 9, the result differences between these three algorithms are visible.

No matter which classifier is applied, we obtain a conclusion that our framework shows high performance in malicious image detection. Moreover, it represents that BPNs performs better than the other two algorithms, and we believe that BPNs is more suitable for our framework. The back-propagation network classifier has the following advantages:

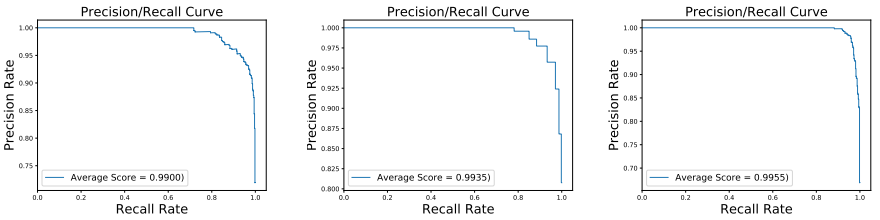
- It can handle thousands of input features without feature deletion.
- It points out the important potential features for classification.
- It performs an internal unbiased estimate of the generalization error.

We also investigate the time consumption of each algorithm by measuring the time to process the entire training dataset and the time to predict 1000 images with malicious payloads. The result is illustrated in Table 2.




(a) LDA (b) RF (c) BPNs

**Fig. 8.** ROC and AUC of algorithms.




(a) LDA (b) RF (c) BPNs

**Fig. 9.** Precision/Recall Curve of algorithms.

### 5.3 Discussion

**Limitation.** We manually inspected some incorrectly classified images and identified the following issues that cannot be proceeded by IMShell-Dec.

- If a malicious image contains a sizeable pure color area, IMShell-Dec may predict the image as the wrong label.
- IMShell-Dec may incorrectly report a benign image with inferior quality.
- If the image path is deliberately obfuscated, our proposed scheme is unable to locate the image's position.

**Future Work.** To address the limitation mentioned above, we plan to find new feature extraction methods to solve the problem that some edges of the image may extract inaccurate features. Besides, we decide to design a better pattern matching method to locate the links of images more accurately.

## 6 Related Work

### 6.1 Malicious PowerShell Script Detection

Several works [8,11,6,16] has proposed their methods and algorithms to detect malicious scripts.

For example, Hendler *et al.* [8] extract features from malicious PowerShell scripts through the bag-of-words model, a natural language processing approach, where the system transform PowerShell commands into a multi-set of words, then calculate their frequency to generate the feature vectors. These feature vectors are further processed with Convolutional Neural Networks(CNNs) and Recurrent Neural Networks(RNNs) to identify the category of PowerShell commands.

Khan *et al.* [11] extract critical features through the wrapper approach to detect unseen malicious scripts. They collect malicious JavaScript codes from client sides, apply the wrapper method to distill an info-enriched feature subset, then feed this feature subset into the detection model. In this work, the author compared four supervised machine learning classifiers (Naive Bayes, Support Vector Machines, K-Nearest Neighbour and Decision Trees), and choose the one with the best prediction performance as the detection model.

Although these research apply different feature extraction strategies, none of them consider the attack vectors outside of the script. Therefore, existing script detection scheme can not identify our proposed attack, as the true attack payload is located in an external resource, and the release script itself is clean and harmless.

### 6.2 Steganography Image Detection

Due to the data structure of image, researchers has proposed several machine-learning based detection method [21,22,23,9,14] to recognize image processed with steganography tools. Wu *et al.* [21] leverage the residual network[7] to detect steganographic images. Ye *et al.* [22] promote a CNNs architecture to analyze steganography consisted of diverse activation modules. Ke *et al.* [9] proposed a hybrid deep learning framework, which combines the bottom hand-crafted convolutional kernels and threshold quantizers pairing with the upper compact deep-learning model.

For the adaptive pattern-based detection, Chen *et al.* [4] utilize local texture pattern (LTP) to detect binary image steganography, which LTP describes the texture distribution of areas and consist of pixels within the areas. Similarly, a feature selection approach [2] implemented adaptive inertia weight-based particle swarm optimization is proposed. Saman *et al.* [18] proposes a novel blind statistical analysis technique to detect the least significant bit flipping image steganography.

## 7 Conclusion

We investigate a new class of PowerShell attack combined with steganography, which allows an attacker to conceal their malicious payload in a medium outside of script, thus bypassing conventional intrusion detection methods. To examine the feasibility, we generate images hosted with script through a popular steganography tool, *Invoke-PSImage*, then retrieved and executed the payload

successfully through another harmless release script. Pilot research shows that the synthesized image has no visual difference with the original, and multiple mainstream defenders failed to intercept the image nor the release script. Both results confirmed the stealthiness of this attack.

To address the emerging threat, in this paper, we propose a machine-learning-based defense framework, IMShell-Dec, to identify malicious PowerShell script that hiding their real payload in the external image. We train and evaluate our proposed framework on a synthesized dataset, in which our framework achieved high detection performance across multiple measurements. Our work can serve as an inspiration in designing a more robust and secure detection model against the proposed attack schemes.

**Acknowledgments.** This research has received funding from The Key Program of NSFC Grant (U140525), The National Nature Science Foundation of China (No.61672415), Key Research and Development Program of Shaanxi Province (No.2018ZDCXL-G-9-5), Open Foundation of Science and Technology on Communication Networks Laboratory (No.SXX18641X024), Key R&D Program of Shaanxi Province (No. 2019ZDLGY12-04), Scientific and Technical Innovation Plan of Shaanxi Province (No.201809168CX9JC10) and National Key R&D Program of China(No.2017YFB0801805).

## References

1. Abadi, M., Xie, Y., Yu, F., John, J.P.: Identifying malicious queries (Jul 23 2013), uS Patent 8,495,742
2. Adeli, A., Broumandnia, A.: Image steganalysis using improved particle swarm optimization based feature selection. Applied Intelligence **48**(6), 1609–1622 (2018)
3. Antoniol, G., Ayari, K., Di Penta, M., Khomh, F., Guéhéneuc, Y.G.: Is it a bug or an enhancement?: a text-based approach to classify change requests. In: CASCON. vol. 8, pp. 304–318 (2008)
4. Chen, J., Lu, W., Fang, Y., Liu, X., Yeung, Y., Xue, Y.: Binary image steganalysis based on local texture pattern. Journal of Visual Communication and Image Representation **55**, 149 – 156 (2018)
5. Christodorescu, M., Jha, S.: Static analysis of executables to detect malicious patterns. Tech. rep., WISCONSIN UNIV-MADISON DEPT OF COMPUTER SCIENCES (2006)
6. Fass, A., Krawczyk, R.P., Backes, M., Stock, B.: Jast: Fully syntactic detection of malicious (obfuscated) javascript. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 303–325. Springer (2018)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
8. Hendler, D., Kels, S., Rubin, A.: Detecting malicious powershell commands using deep neural networks. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security. pp. 187–197. ACM (2018)
9. Ke, Q., Ming, L.D., Daxing, Z.: Image steganalysis via multi-column convolutional neural network. In: 2018 14th IEEE International Conference on Signal Processing. pp. 550–553 (2018)

10. Kertesz, V., Whitehead, D., Burke, J., Tanna, H., Garg, S., Rao, A.R., Lakshmi, S., Mishra, S., Lakshminarayana, J., Tamanna, P.K., et al.: Dynamic data exchange server (1998), uS Patent 5,764,155
11. Khan, N., Abdullah, J., Khan, A.S.: Defending malicious script attacks using machine learning classifiers. Wireless Communications and Mobile Computing **2017** (2017)
12. Lee, T., Mitschke, K., Schill, M.E., Tanasovski, T.: Windows PowerShell 2.0 Bible, vol. 725. John Wiley & Sons (2011)
13. Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering **34**(4), 485–496 (2008)
14. Li, B., Wei, W., Ferreira, A., Tan, S.: Rest-net: Diverse activation modules and parallel subnets-based cnn for spatial image steganalysis. IEEE Signal Processing Letters **25**(5), 650–654 (2018)
15. Li, Z., Chen, Q.A., Xiong, C., Chen, Y., Zhu, T., Yang, H.: Effective and lightweight deobfuscation and semantic-aware attack detection for powershell scripts. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1831–1847. ACM (2019)
16. Milosevic, J., Sklavos, N., Koutsikou, K.: Malware in iot software and hardware. In: Workshop on Trustworthy Manufacturing and Utilization of Secure Devices. pp. 14–16 (2016)
17. Moser, R., Pedrycz, W., Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th international conference on Software engineering. pp. 181–190. ACM (2008)
18. Shojae Chaeikar, S., Zamani, M., Abdul Manaf, A.B., Zeki, A.M.: Psw statistical lsb image steganalysis. Multimedia Tools and Applications **77**(1), 805–835 (2018)
19. Ugarte, D., Maiorca, D., Cara, F., Giacinto, G.: Powerdrive: Accurate deobfuscation and analysis of powershell malware. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 240–259. Springer (2019)
20. Wilson, E.: Windows PowerShell 3.0 First Steps. Pearson Education (2013)
21. Wu, S., Zhong, S., Liu, Y.: Deep residual learning for image steganalysis. Multimedia Tools and Applications **77**(9), 10437–10453 (2018)
22. Ye, J., Ni, J., Yi, Y.: Deep learning hierarchical representations for image steganalysis. IEEE Transactions on Information Forensics and Security **12**(11), 2545–2557 (Nov 2017)
23. Zeng, J., Tan, S., Li, B., Huang, J.: Large-scale jpeg image steganalysis using hybrid deep-learning framework. IEEE Transactions on Information Forensics and Security **13**(5), 1200–1214 (May 2018)