

Generation and Evaluation of Reliable Dataset for PowerShell Based Fileless Malware Detection: A Machine Learning Approach

Manish Kumar Meher¹, Mayank Kumar¹, Adyasha Rath¹, Ganapati Panda²

¹Department of Computer Science and Engineering

²Department of Electronics and Communication Engineering
C. V. Raman Global University, Bhubaneswar – 752054, INDIA

Corresponding email: ganapati.panda@gmail.com

ABSTRACT:

In windows environment PowerShell becomes an integral part who's responsible for system automation, management and configuration tasks. It is very flexible and helps in performing all the operations, which makes it more popular among developers and users. Yet, this flexibility is also the reason for being targeted for criminal activities by implanting PowerShell scripts in the form of File less malware as it doesn't leave any trace in memory. So detecting these scripts before execution is the best way to counter the fileless attacks. For that, an authentic and reliable dataset with advanced Machine Learning (ML) and Deep Learning (DL) models was required with the most useful feature sets for better detection. This paper implemented a generalized workflow to create a reliable dataset by collecting PowerShell scripts from different repositories and cleaning the obfuscated script by implementing a deobfuscation technique. Then evaluated them with a well-known antivirus (AV) platform, Virus Total, to label those scripts. The extracted meaningful features from those clean and labelled scripts are passed as inputs to ML models for evaluation of those models. Among defined models, Trigonometric Functional Link Artificial Neural Network (TFLANN) performed outstandingly with an accuracy of 97.68%. The proposed reliable dataset offers effective support to future research in PowerShell-based file less malware detection and can be used to create more effective machine learning-based cybersecurity mechanisms.

Keywords: PowerShell, Fileless Malware, Deobfuscation, RBFNN, Machine Learning.

I. INTRODUCTION

Cyberattacks that have been increasingly advanced through the use of scripting software like Microsoft PowerShell have become a major challenge to the contemporary security architecture. As a valid administrative and automation platform that is part of the Windows platform, PowerShell provides extensive access to the operating system, which attackers frequently use to introduce, install, and manage malicious code. The portability and the malleable character of PowerShell allow assailants to develop very obfuscated codes that can circumvent the

standard static detection devices and endpoint defences. Therefore, creating efficient mechanisms for analysing and identifying malicious uses of PowerShell has become a pivotal research topic in the sphere of cybersecurity and malware analysis.

One of the biggest challenges when it comes to the detection of PowerShell-based attacks is the vast amount of obfuscation and encoding that it makes use of. Attackers frequently implement several levels of transformation: Base64, hexadecimal, character replacement, and compression to hide their malicious activities from the analysers that are not dynamic and from the human eye. These scripts are able to dynamically decode, decompress, or rebuild their payloads at runtime, thus making reverse engineering and feature extraction more difficult. Consequently, productive and effective deobfuscation is a precondition to any data-driven analysis, such as machine-learning-based malware classification and behavior profiling.

Many existing literatures often considers the partially validated or synthetically generated datasets. This may hamper the performance and detection accuracy with real-world application machines. So this study focuses on validating the PowerShell scripts with a reliable and publicly admissible antivirus platform for labelling. With this, a reliable VirusTotal-verified dataset with admissible malware and benign sample labelling in the dataset. To deobfuscate the script, the study applies common deobfuscation techniques and a static feature extraction technique to extract features from the clean script. As neural networks were employed to effective capture of complex patterns which enhance detection of malware. TFLANN and Radial Basis Function Neural Network (RBFNN) classifiers were used, which significantly improve dataset reliability and reliable classification in PowerShell-based fileless malware detection.

II. LITERATURE SURVEY

In paper [1], the authors propose a deep learning-based method where eXtreme Gradient Boosting (XGBoost) is used for classification and a Stacked Denoising Auto-Encoder (SdA) for feature extraction to detect malicious PowerShell scripts. The paper highlights the difficulty of dealing with severely obfuscated PowerShell malware and uses SdA to learn robust representations automatically on noisy and high-dimensional script data. The authors

generate labeled and unlabeled datasets of obfuscated benign and malicious samples, thus allowing supervised and semi-supervised training. Experimental findings demonstrate good performance whereby the supervised model has a 98% True Positive Rate (TPR) and a 0.6% False Positive Rate (FPR), which is superior to classical Support Vector Machine (SVM) and Naive Bayes models. Nevertheless, such weaknesses as reliance on fixed-length American Standard Code for Information Interchange (ASCII) representations and poorer performance in semi-supervised mode exist, and further research into dataset expansion and multi-language detection is recommended.

The paper [2] presents a defense framework called MPSD, which targets the identification of malicious PowerShell scripts by performing both a static and a behavioral analysis. The authors create a labeled benign and malicious PowerShell sample set and preprocess it to address the encoding, obfuscation, and script normalization processes. MPSD incorporates feature engineering such as command patterns, entropy, Application Programming Interface (API) usage, and behavioral indicators and then machine-learning classifiers to distinguish between attack scripts and legitimate automation. Experiments show good performance, high accuracy, and fewer false positives with the different threat situations. Nevertheless, it is weak at detecting highly obfuscated or multi-stage payloads, which means that it does not have the generalization in emerging families of malware. Future enhancement entails dynamic analysis incorporation and a more profound semantic modeling to achieve powerful detection.

PowerDP [3] is a hybrid architecture suggested to decide on combining deep learning with program analysis to automatically de-obfuscate and behaviorally profile malicious PowerShell commands. The model, as illustrated in the system overview, initially predicts obfuscation types with character-distribution features, followed by a regex-based deobfuscator, and lastly with multi-label behavioral classification of Abstract Syntax Tree (AST)-based vector representations. It consists of a dataset of 3,057 real-world malicious scripts and more than 36,000 variants that have been artificially obfuscated. According to performance results, the classification of obfuscation and behavioral profiling are characterized by an excellent rate of accuracy (99.82 and 98.53, respectively). Weaknesses consist of the inability to work with multi-layer obfuscation, which is hard to handle, and the need to have manually generated ground-truth labels. The development of future work will be to increase the number of behavioral categories and enhance semantic recovery.

The paper [4] is a comparative analysis of classical Machine Learning (ML) models and quantum kernel-powered models in detecting malware in embedded systems supply chains. The authors run SVC, SVC with quantum kernels, and Quantum SVC with the ClaMP dataset (5210 samples, 108 features). The findings demonstrate the quantum-enhanced models reduce multicollinearity, and in low-dimensional feature space, high precision is obtained, compared with the robustness of classical SVC. Although it is more accurate, the time spent executing on real quantum hardware is much more (approximately 1900 seconds compared to 0.15 seconds), and the existing Noisy Intermediate-Scale Quantum (NISQ)-era limitations are visible. In conclusion, the paper found that Qt Modeling Language (QML) has potential in the future as a means of cybersecurity but has noise issues in hardware, decoherence, and scalability issues.

The article [5] presents four variants of RBFNN that can be modified to enhance the rate of convergence, accuracy, and generalization when performing nonlinear classification and modeling. The models include additions like adaptive input-layer weights, exponential input transformations, and learnable Gaussian widths; the proposed Model M-6 is the best performing in all the XOR classifications, nonlinear system identifications, and channel equalizations. Dramatic improvements in root mean square error and training steps over traditional RBFNNs (M-1 and M-2). Although higher accuracy is a good indication, higher cost per iteration shows that more complex models require higher computation cost. The authors mention the prospects in optimization-based tuning and expansion of use to control systems.

This key contribution of the paper presents a reliable PowerShell-based fileless malware detection framework with four major contributions. The first contribution of the paper is creation of a reliable dataset with proper validation and labeling. It has to be done by collecting PowerShell scripts from various sources and VirusTotal validation for labeling. Second, a structured deobfuscation pipeline prepared to gain the structural and semantic characteristics from the obfuscated script. Third, the study implies a static feature extraction to extract from the clean scripts to capture diverse statistical features. Finally, the effectiveness of the prepared dataset with feature set is tested by evaluating the TFLANN, RBFNN, and SVM classifiers.

Fig. 1 represents the entire workflow of the paper, which consists of the collection of reliable samples, cleaning of those samples for better validation and labeling, meaning feature extraction, and use of reliable ML models to test the dataset impact as well as model performance.

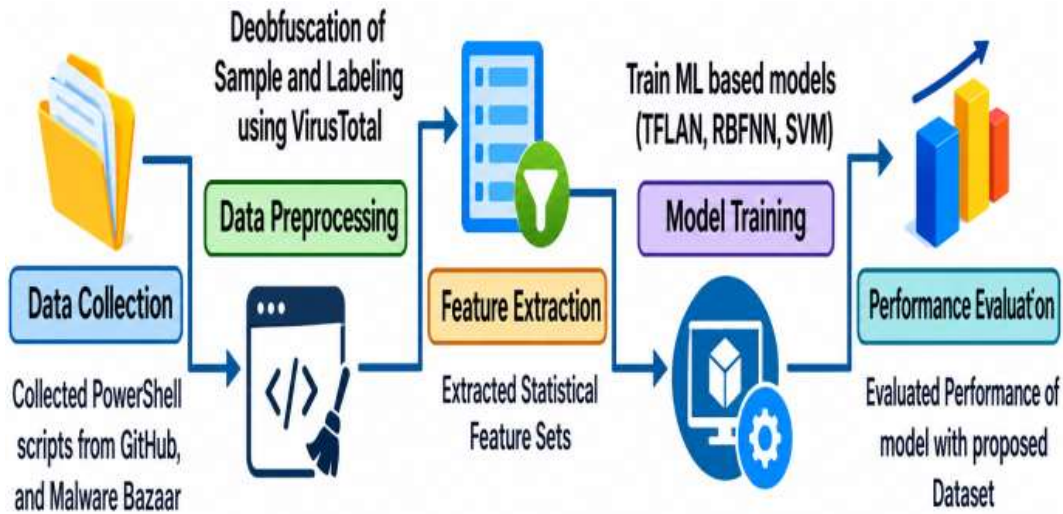


Fig. 1. Overall Workflow of the Proposed ML-Based PowerShell Malware Detection Pipeline

III. DATASET

A. Script Collection

More than 4,000 PowerShell script samples were collected from various open sources, such as GitHub [6], [7] and Malware Bazaar [8] repositories, to obtain a reliable and analytically enriched dataset to train and evaluate machine learning models. The multi-source collection approach was necessary to provide a wide range of diversity, including benign administrative scripts, security testing tools, and a wide range of real-world malicious PowerShell threats. This kind of diversity is indispensable to reduce bias in the dataset and improve the model's generalizability.

B. Script Deobfuscation

Since most malicious scripts use obfuscation layers to hide malicious intention, the scripts were processed through deep deobfuscation right after collection. This involved the decoding of Base64 and hexadecimal encodings, the de-compression of compressed payloads, de-aliasing PowerShell, unfolding concatenation patterns of strings and formatting normalization [9]. It was not only aimed at revealing the lurking logic but also to bring together heterogeneous scripts to the form to which they could be converted in a reliable manner to extract features and analyze behaviors. The deobfuscation pipeline represents several steps to ensure the scripts were clean and feasible to extract meaningful features.

This first step of the pipeline contains the normalization phase which include surface cleaning or structural sanitizing. It mostly removes null spaces between words which disrupts the analysis. It also scores out the escape quotes such as the ones that look like; ', and empty ones such as; ', ' ' etc, to clean the script. It assists in re-finding the fundamental form of

script, which has an impact either in tokenization or character visibility. The second stage was Semantic Recovery, i.e. symbol and escape normalization, which was meant to cover up the deformed command. The step eliminates a tick (') , an escape or line continuation command in the PowerShell script. It also removes the symbol of caret and the symbol of string join (+) and this aids in restoring the sequence of the command. The concealed command names, Uniform Resource Locators (URL), or coded parameters are revealed in effect of this. The logical expansion was carried out in the third phase where the symbolic toned down representations of the commands were substituted by their real and explicit equivalents such as iex -> Invoke-Expression, iwr-> Invoke-WebRequest, etc. This is because, this stage verifies the entire semantic meaning of the script which is far more readable.

The second step performs the decoding of encoded strings by applying two decoding techniques, i.e., Base64 Decoding and Hexadecimal Decoding. These decoding techniques are the most common and frequent to use in script obfuscation. The Base64 decoding phase aims to extract the command, payloads, and URLs. These are encoded using Base64 Encoder, which helps to avoid signature-based detection mechanisms. Tracing the regular expression pattern of [A-Z, a-z, 0-9, +/=]{30,} in this particular stage and considering a string length of a multiple of four for further decoding. Then the strip ASCII () function was used in the considered string to extract the actual command or payload from the encoded string. Hexadecimal Decoding was like Base64 encoding; the hexadecimal encoding used an encoding pattern to encode the ASCII characters. So this module locates the sequence of hexadecimal values by using the (0x[0-9a-fA-F]{2}[,]*){4,} pattern. Among the set of converted hexadecimal values, identify and convert from hexadecimal

notation to raw bytes and then finally print it to readable ASCII text.

The third step of the pipeline reintegrates the output of the above two steps and stores it in a newly created file with the .ps1 file extension. This file had the same name as the original file and was renamed with a

prefix of “d_” to mark it as deobfuscated. All the deobfuscated script files were stored in a separate file for script validation with the VirusTotal antivirus portal.

Obfuscated PowerShell Script

```
("eX([SysteM.TEXT.enCOding)::UtIF8.getsTRING((SysteM.CoNVERT))::FromBase64STRing
("ZmVuYSRpb2 4g 20'FzQK VT 1WNbGbg'4Y2jYTKv W8pnZ8'9n WNN1 R3F 'luXp S'ZU9 GdW44cUB1c
wogTC'AgliCagc'LA g I3 CBUQ21kbic'W0 Qm1uZGLuZyhT Z'Hx "w0X'Jp100g")2h0d iF BcD18vc'If
WscAF'c2'XdvCm8uB3nL2RvY3"VZ1dF 56YS"Rpb24v 1f2 9t0wVg' Hc Wh "Gwhp0.0ng HwwM48H9 1w0yc
y0} Vw" 2h0w7 1qs'a71u2nY"K3 L 6T4'2F8Fk 0c' HeltCgKICa0TCag 1cEB0'UGFy "WU1lds" yx
"Et1hbHsdSgYsa70" IC"8"Uc n" W0"KT Gag c"C" BcCa 1l1dGvWl' k'1Kb1F H Z0lIFtB IQV" Woub0
EV91W1sTg(3 3' JUF R'XIE9 j510k1" Cag1C' Aad" 2N5SkBUwIx0P @ 2'5s6' JA gU0AG'g1C' A g1C
"AqT' F' Oya" he W0 2"kT Q" H W" ZcF08' 3'J10H0QH k' rc" H0p",MVK4 1qzT" T(V0 1" gY 1w0q
"SsY5" WnD1 MF"8nTN0TWVG00" QcFN0ck2QW5" 1ZQvdClgTCAg1C' A" k0U9ERS4910cD" 80'dk" RV "1W" K
"AC" A6" IC" Ag1A" 0 g"Cag3C8088" Na" c8nZK" 0M2u0sGFS" YW" 'If dGV' 5' LnR" pTUV2cEfFuFsF" y0U'
11VEV SX00gt'C' Ag1C' RD' Q8N' CR0A' R1C" 4FHw" Wst00q" Hoq' LcAqLc' W7A" Qq1ccCq' 18nY0ba n' c'3XoK
"0F' Ftp' T1h3M80004" D5E" gv7NlKE" F 31TK' U' unu' 4x2ct0n" v521' V' m d' V3' WdH6g8S20a8XXK5M25i0HF f50nf5
42" E' PmUG" 3IGz23bWnK" W 1LaT Z0V" NoLXZ' HsGxwb1h0MMB:3M0J3bon' q' N M F" X0H" hCKR" hCkZ7IGJ9MqS20K"
o4n4e' s3W' Tm2n' nW 1'8'jeTame' QmA3' oM Bc Irt XQ' x' 1' NIJbLRE95J3n6c UFZm0x' Zbv2" Din BeT879QJ00Uu0n
"83' W' Z00" HqN14gy59' U5' 60' 5" W08J1RvcXTB10E" d' Sa0JlavM8J35Tren14S0' VJvI' Fk0n' BQHM4InoppRifawH00
"0a" NyqM"adzyWc" vU" W' 6" d' Mp' aa A" v' 51WwZ0p' Tux0' B' 5" N9L0+vm+H58J" b0V' H9y50' 93' 0uit' Y0QxZ'
CT2Dq' 2m0IF' BTRnAn0h' V3b 3mLIR' NYKVA'cG' Fuc21vb1' T' W3' p' ch0B0ZS0YNLU'Z0kIACE10C2K'K' "QW' 8e' 0 NaUHY)
```

Deobfuscated PowerShell Script

```
function gAYAuHXsFnjbccebCOWJg9ogZseGquQzRe0Fugxy0
{
    [CmdletBinding(HelpUri = 'https://psframework.org/documentation/commands/PSFramework/')]
    Param (
        [Parameter(Mandatory = $True)]
        [System.Management.Automation.ScriptBlock]
        $ScriptBlock,
        [Parameter(Mandatory = $True)]
        [string]
        $Name,
        [PSFramework.TablexPaNsION.TePpScRiPTmode]
        $Mode = 'Auto',
        [PSFramework.paRamEter.tIMEsPaNParAMETER]
        $CachedurAtion = 0
    )
}
```

Fig 2: Comparison of Obfuscated and Deobfuscated PowerShell Code after applying deobfuscation process.

The Fig:2 represent output of deobfuscation process. It presents the importance and how the deobfuscation process enhances the PowerShell script clarity and quality. This process crucially effects the feature extraction which leads to increase ML based malware detection models performance.

C. Script Validation

As many of the research articles lack the validation of script in an admissible authentication manner. After deobfuscation, VirusTotal [10] validation of every script was performed, with a view to proper and reliable labeling. As labeling the PowerShell scripts was very critical because this might affect the overall performance of the model in real-life applications. The lack of proper labeling leads to a high rate of false positives and true negatives. This is a very concerning area in malware analysis because the whole idea of the work failed. The Virus Total API was run with script hashes and the profile of the detection as the number of Anti-Virus (AV) engines

that detected the sample and the associated threat levels were captured. Each AV engine follows a closer inspection of each script and tries to find the intention behind the script. This cross-check was used as a quality assurance tool, which filtered out the misclassified samples and increased the data authenticity, which plays a critical role in malware analysis. Benign script samples were not flagged by any of the more than 60 AV engines.

IV. FATURE EXTRACTION

The concept of feature extraction states that obtained specific characteristics from the raw samples are called features. These features are extracted, which can be used for further analysis, detection, or classification by training ML-based models. When dealing with suspicious and malicious samples in cybersecurity, these extracted features from raw data prove beneficial in discovering the presence of patterns and associations between the normal and malicious files.

The aspect of feature extraction in the sample plays a critical role in the ML-based classification since the sample is what is trained, and the model trains on the presented feature in the form of the vector. In the

detection of malicious PowerShell scripts, it is more important to select the effective features in order to obtain

Benign Script file Report						
Filename	SHA256	MD5	SHA1	Flags Detected	Detected By	Behavior Info
Ben1_100.ps1	d34d17029d4cd489b00c7196ed13d30a377e4639	23d6960f11d6666bed	23d6960f11d6666bed	0	None	N/A
Ben1_108.ps1	3798f39130564cde3890035b229cfb78c5692	1f2932c8c4876d47c55f	1f2932c8c4876d47c55f	0	None	N/A
Ben1_113.ps1	0cc871c0b35e4df7584d0c98e4a7ed3478319c2	e5758ebca3b27a7f08b	e5758ebca3b27a7f08b	0	None	N/A
Ben1_119.ps1	3f6258764f20e1673098bcb5ca254c0f098c72a1	588fe46af4a782f8a8e31	588fe46af4a782f8a8e31	0	None	N/A
Ben1_14.ps1	7c53dfb087131019d1b15327e38516029723be	bb33d079f19349f3c22a	bb33d079f19349f3c22a	0	None	N/A
Ben1_142.ps1	2b35b6ca6d369eba36f2f49b0adf2e1e9e7b6c5	f2f4e90b4d48e0d0e5	17fb6f41b4ce1f1f0c8	0	None	N/A
Ben1_162.ps1	8e3bec6adcd0a13c8d66a9d72d7efa66cc4b6	6d8a17e042b52da6d4f	6bcb33e58cd9d9f22a	0	None	N/A
Ben1_3.ps1	8981a3fba253b794132407d62cf9725928979a	76d76e6c5e41f990ebc0	76d76e6c5e41f990ebc0	0	None	N/A

Malicious Script file Report						
Filename	SHA256	MD5	SHA1	s Date	Detected By	Behavior Info
Ben1_10.ps1	af6f181e8993f6f018a5	93f6f018a5	2eca5a01be	7	MicroWorld-eScan, VIPRE, BitDefender, Emsisoft, GData, Arcabit, VBA32	pantera
Ben1_101.ps1	70c6c4028f29c1d46e4a	29c1d46e4a	5f55854a61f	7	MicroWorld-eScan, Arcabit, BitDefender, Emsisoft, VIPRE, GData, VBA32	pantera
Ben1_104.ps1	ac4e775f529b07ef6e43e	b07ef6e43e	96709c0423	11	MicroWorld-eScan, CTX, Symantec, BitDefender, Emsisoft, VIPRE, GData, Google, Arcabit, Vanist, VBA32	pantera
Ben1_105.ps1	896e48c840e51d1a8ad	6e51d1a8ad	adf2e9f6895	7	MicroWorld-eScan, BitDefender, Emsisoft, VIPRE, Arcabit, GData, VBA32	pantera
Ben1_106.ps1	60dc31008d6756e84c85	6756e84c85	61102562e6	7	MicroWorld-eScan, VIPRE, BitDefender, Emsisoft, GData, Arcabit, VBA32	pantera

Fig 3: Overview of VirusTotal Generated Report with Hash Values, name and count of detected VM along with behavior info.

After careful examination of the script by all the AV engines generated a detailed report containing fields like filename and hash values, along with the number of flags detected, the AV engine name, and the behavior type of that particular script. Fig. 3 represents the overview of the report generated after the validation process of all available scripts.

TABLE I

Details of PowerShell Script Samples

Category	Samples	Label
Malicious	1037	1
Benign	1013	0

Table. I represent the total number of malicious and benign samples that were selected by considering the report generated in this process. Here the malicious script samples were flagged by at least 7 AV engines, and the best dataset. It causes the highest detection accuracy or fewer true negatives. Just like in the instance of malicious PowerShell scripts, it is important that malicious scripts must not go through the security system without raising a flag [11].

Once the deobfuscation is done, it enables extraction of features from the samples more effectively and precisely, making the model more effective in terms of training. To make it possible Static Feature extraction from PowerShell script carried out, which possess the features mentioned;

The Bypass Features define the features present in the script responsible for the background run of the

script, avoiding firewall detection or tampering with security controls. Indicators like bypass, hidden or minimized, proxy, and uninstalling or Assembly are commonly used during feature extraction Microsoft DevBlogs.

In the category of Downloader Features responsible for scripts performing the operation to fetch external malicious payloads or files or scripts into the system over the network. The Download File, Download String, Use Default Credentials, Read To End, Decompress, Memory Stream, and System.Net. Web Clients are some of the most used features.

The Injector feature identifies the component responsible for injecting malicious code or script into memory, which refers to fileless malware types. Features like Add-Type, Win32Functions, kernel32.dll, Virtual Alloc, Create Thread, Get Bytes, and ToBase64String are used in code injection for fileless attacks.

In the Payload Feature category, responsible for decoding, assembling, or executing the malicious logic or malicious file downloaded into the system over the network. The Payload Feature is like FromBase64String and System. Reflection, Get Type, Set-Strict Mode, start-job, receive-job, and get proc address are the most common among all.

The Task Execution Features represent the operations at the system level, generating child processes or mostly executing commands in the PowerShell. Windows Style, CreateNoWindow, System. Diagnostics. ProcessStartInfo, Wait-Job, Get-Process, and Add are the most used features.

The Execution/General Features reveals the execution mechanism and script behavior to identify obfuscation or command invocation to identify malicious scripts. New-Object, Invoke-Expression/IEX, cmd.exe, powershell.exe, and Replace() and \$env are the general features.

Features like static code metrics define the script length, number of lines, entropy, number of URLs, and so on to find the pattern. script_length, num_lines, avg_line_length, entropy, is_one_liner, num_urls, and num_ips are some basic and most used features for the analysis. All the above category features have been used in the process of feature extraction from the .ps1 file.

V. RESULTS AND DISCUSSIONS

For the detection, the PowerShell malicious script TFLANN, RBFNN and SVM models are used. The TFLANN was a lightweight neural architecture capable of capturing nonlinear malicious patterns. The RBFNN uses a radial activation function to capture clusters of malware behaviors by modeling localized cluster patterns, which is effective for malware behavioral features. SVM depends on a subset of informative support vectors instead of the entire dataset so it was effective in case of small dataset. It also provides kernel-based learning which helps to capture nonlinear patterns in highly obfuscated scripts.

For the mathematical formulation of the model TFLANN Let the extracted feature vector be represented as

$$x = [x_1, x_2, \dots, x_n] \quad (1)$$

Where each element corresponds to a static feature derived from the PowerShell script.

To model nonlinear relationships, a trigonometric functional expansion is applied:

$$\begin{aligned} \varphi(x) = & [w_1^{(x)} \odot x_1, \dots, w_N^{(x)} \odot x_N, \\ & w_1^{(\sin)} \odot \sin(x_1), \dots, w_N^{(\sin)} \odot \sin(x_N), \\ & w_1^{(\cos)} \odot \cos(x_1), \dots, w_N^{(\cos)} \odot \cos(x_N)] \quad (2) \end{aligned}$$

Where $\odot$ denotes element-wise multiplication.

The Softmax activation function computes class probabilities:

$$\hat{y}_k = \frac{e^{\theta_k(x)}}{\sum_{j=1}^C e^{\theta_j(x)}}, \quad k=1,2,\dots,C \quad (3)$$

The categorical cross-entropy loss function is defined as

$$\mathcal{L} = - \sum_{k=1}^C y_k \log(\hat{y}_k) \quad (4)$$

The end-to-end forward computation is expressed as

$$\hat{y} = \text{Softmax} \left(\begin{bmatrix} w^{(x)} \odot x \\ w^{(\sin)} \odot \sin(x) \\ w^{(\cos)} \odot \cos(x) \end{bmatrix} \right) \quad (5)$$

Table II shows the performance comparison of TFLANN, RBFNN and SVM models carried out under 80-20 and 70-30 training-testing splits. This displays the clear distinctions in convergence behavior, classification accuracy, and overall multi-criteria ranking of the used models. In all metrics, TFLANN performs better than the other classifiers with the best accuracy (97.68%), F1 score (0.9768), and AUC (0.9918) in the 80 -20 setting. Its TOPSIS score of 1.0 makes it the best-performing model, which proves that the model has a high generalization ability and can work well even in the presence of nonlinear PowerShell malware characteristics.

The given model was tested on a feature based dataset of 2,050 samples and 76 features. The three-order trigonometric expansion was used to improve the non-linear representation, and this led to 228 input neurons (76 x 3). An 80:20 and 70:30 train test split was used to divide the dataset such that 80 and 70 percent of the sample was used in the training set and 20 and 30 percent left in the test set respectively. The training was done for 100 epochs and with the Adam optimizer, the learning rate is set to 0.001, which was selected as this provides a stable and efficient convergence. At the output layer, a Softmax activation function was used to facilitate the binary-class statistical identification

TABLE II

COMPARISON OF RANK AND OTHER PERFORMANCE METRICES OF DIFFERENT MODELS

MODEL	ACCURACY	F1 SCORE	AUC SCORE	TOPSIS SCORE	RANK
TFLANN 80-20	97.68	0.9768	0.9918	0.976353	1
RBFNN 80-20	96.59	0.9650	0.9964	0.915398	2
TFLANN 70-30	96.52	0.9651	0.9890	0.904211	3
RBFNN 70-30	92.20	0.9164	0.9040	0.405598	4
SVM 80-20	90.85	0.9075	0.9223	0.365556	5
SVM70-30	86.69	0.8543	0.9222	0.089992	6

The TFLANN 70-30 version is also competitive with RBFNN 80-20 by standing at the third position. RBFNN models show moderate performance whereby the 80-20 split shows an accuracy of 96.59% with a significant drop in accuracy below the 70-30

split, signifying that the models are sensitive to smaller amounts of training data. The SVM 80-20 split performs better the 70-30 with an accuracy of 90.85%. Their positions in the TOPSIS ranking make them dependable yet weaker options.

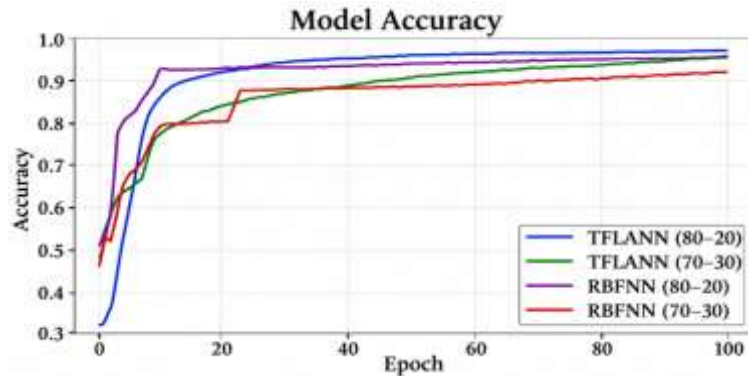


Fig. 4. Comparison of different models variation of accuracy during Trans phase

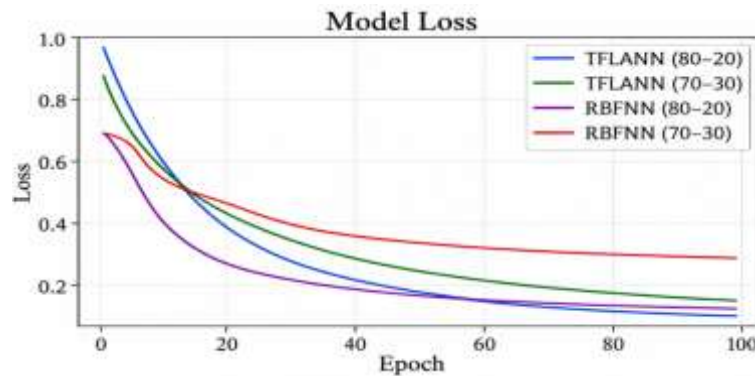


Fig. 5. Comparison of loss plots during training phase of different models

In general, the assessment has reached the final verdict that TFLANN is the most useful and stable model in terms of detecting malware through PowerShell, followed by RBFNN then SVM. Fig. 4 represents that the TFLANN was the fastest and most stable with accuracy and convergence. In both data splits, TFLANN outperforms RBFNN and achieves nearly perfect accuracy within the early epoch with the 80-20 data splits. Fig. 5 shows that in the context of maintaining lower training loss across the epochs, TFLANN performs better than RBFNN. It highlights its superior optimization efficiency and ability to model nonlinear PowerShell features effectively.

The analysis of errors shows that the misclassification of samples is mostly because of highly obfuscated PowerShell scripts, particularly the use of multilayered Base64 encoding schemes, XOR-based payload concealment, typical concatenation string schemes, or any unknown encryption. Even though the proposed deobfuscation module is applicable in normalizing the majority of the scripts, some samples have highly obfuscated or half-obfuscated scripts that yield semi-recovery. Consequently, the extracted

static features, especially the lexical and structural patterns, are not entirely indicative of the actual behavior of the script, and this is the cause of classification errors in all the models.

RBFNN has a stronger degradation at the 70-30 split since its radial basis framework is much dependent on the concentration and the coverage of the training samples. It has more difficulties in positioning its centers correctly in the feature space, leading to poor generalization to less popular or highly obfuscated malicious examples with fewer training examples. RBFNN also is susceptible to noisy features; partial deobfuscation enhances the impact and increases the performance disparity in the case of limited training data. Conversely, TFLANN is more resilient because of its functional expansion and more adaptive nonlinear modeling, which enable it to represent higher-order feature interactions more effectively in the face of a degraded semantic signal.

VI. CONCLUSION

To ensure the reliability of the dataset in the detection of PowerShell-based malware, this study addresses

limitations of existing work, which often lacks proper deobfuscation and verification of the script while labeling. The proposed pipeline starts with the script deobfuscation process, which is very crucial in the context of the quality and clarity of scripts, which affects the performance of verification and feature extraction. Following deobfuscation, all samples passed through the VirusTotal validation process, which is a multi-engine AV platform to label those deobfuscated scripts, and then static features were extracted from the scripts. The ML models were trained and tested using the balanced dataset for performance evaluation. The TFLANN model performs outstandingly with a 97.68% accuracy and F1 score in the case of an 80-20 data split.

This research emphasizes to enhance model reliability by constructing reliable dataset validation and static feature extraction in PowerShell based malware detection. In addition, this study shows the effectiveness of neural network-based models in detection of PowerShell based malware detection by focusing on pattern recognition of malicious behaviors. These contributions collectively advance the robustness and practical applicability of machine learning techniques in malware detection.

The next step of work will be the extension of the dataset with the usage of many more and diverse PowerShell samples to enhance the model generalization and its resistance to new malware families. The deobfuscation system will also be expanded to support more sophisticated and multi-layered obfuscation schemes that will allow script semantics to be more accurately recovered. Along with this, investigating more sophisticated methods of feature extraction to enhance the depth and discriminative capabilities of features to be used by the machine learning model training. It is believed that these enhancements can also enhance the detection performance by a great margin and enhance the stability of PowerShell malware classification systems.

REFERENCES:

- [1] Alahmadi, A., Alkhraan, N., & BinSaeedan, W. (2022). MPSAutodetect: a malicious powershell script detection model based on stacked denoising auto-encoder. *Computers & Security*, 116, 102658.
- [2] Wu, M. H., Hsu, F. H., Hunag, J. H., Wang, K., Liu, Y. Y., Chen, J. X., ... & Yang, H. T. (2024). MPSPD: A Robust Defense Mechanism against Malicious PowerShell Scripts in Windows Systems. *Electronics*, 13(18), 3717.
- [3] Tsai, M. H., Lin, C. C., He, Z. G., Yang, W. C., & Lei, C. L. (2022). PowerDP: De-obfuscating and profiling malicious PowerShell commands with multi-label classifiers. *IEEE Access*, 11, 256-270.

- [4] Passo, S. J., Kothavade, V. H., & Prevost, J. J. (2025, June). Supply Chain Malware Detection via Classical and Quantum Kernel Methods in Embedded Systems. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)* (pp. 208-215). IEEE.
- [5] Panda, S., & Panda, G. (2021). On the development and performance evaluation of improved radial basis function neural networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(6), 3873-3884.
- [6] Fleschutz, PowerShell Scripts Collection. [Online]. Available: <https://github.com/fleschutz/PowerShell> accessed on 12/10/2025.
- [7] GhostPack, Invoke-Evasion. [Online]. Available: <https://github.com/ghostpack/invoke-evasion> accessed on 15/10/2025.
- [8] abuse.ch, Malware Bazaar. [Online]. Available: <https://bazaar.abuse.ch/browse/> accessed on 27/09/2025.
- [9] Xiong, C., Li, Z., Chen, Y., Zhu, T., Wang, J., Yang, H., & Ruan, W. (2022). Generic, efficient, and effective deobfuscation and semantic-aware attack detection for PowerShell scripts. *Frontiers of Information Technology & Electronic Engineering*, 23(3), 361-381.
- [10] VirusTotal, VirusTotal File Upload. [Online]. Available: <https://www.virustotal.com/gui/home/upload> accessed on 20/10/2025.
- [11] Microsoft Corporation, About Scripts — PowerShell Documentation. [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_scripts?view=powershell-7.5 accessed on 25/11/2025.

Cite this article as:

Manish, Ganapati, and et. al. "Generation and Evaluation of Reliable Dataset for PowerShell Based Fileless Malware Detection: A Machine Learning Approach", *Proceedings of Applied Energy Systems and Computer Science*, 2025, display as online February 2026. Link: <http://actsoft.org/science/act2025-pro/60-esda2025.pdf>, AOI: 10.100.234512.00041

@Copyright to 'Applied Computer Technology', Kolkata, WB, India. Website: actsoft.org, Email: info@actsoft.org, ISBN: 978-81-985770-9-2