Zero-Shot Reboot-State Detection Across Android Malware Families

Inspired by the research paper:

Detecting New Obfuscated Malware Variants: A Lightweight and Interpretable Machine Learning Approach

Oladipo A. Madamidola, Felix Ngobigha, Adnane Ez-zizi

University of Suffolk, Waterfront Building, IP4 1QJ Ipswich, UK

Abdullah Atahan Türk

N24120222

Introduction

Android’s open ecosystem and rapid app-release cycle have made it the primary target for malware authors, who routinely ship new families or obfuscate existing ones to evade static and session-based sandboxing. Traditional dynamic analysis often runs each sample only once, missing behaviors that only trigger after a device reboot (e.g. persistence routines, payload activation). As tens of thousands of new APKs appear daily and many variants lie dormant until after reboot, there is a critical need for models that can distinguish “pre-reboot” vs. “post-reboot” behavior and, crucially, that generalize to never-before-seen families.

This project first extends the study “Detecting New Obfuscated Malware Variants: A Lightweight and Interpretable Machine Learning Approach” by evaluating gradient-boosted learners for feature selection and classification, improving the results achieved by the previous study. In light of these improvements, it was envisaged that a similar study could be conducted on a different dataset.

The main aim of this project is performing dynamic analysis on Android malware, using a reboot state labeled feature extraction dataset to distinguish pre-reboot and post-reboot behavior while ensuring the models remain lightweight and interpretable.

Related Work

The primary reference for this project is the recent study titled "Detecting New Obfuscated Malware Variants: A Lightweight and Interpretable Machine Learning Approach." The authors addressed the detection of previously unseen (zero-day) malware using memory analysis from the CIC-MalMem-2022 dataset. Initially evaluating various algorithms (Random Forest, Naive Bayes, Logistic Regression, K-Nearest Neighbors, Decision Trees), they identified Random Forest as the most accurate and interpretable. Training exclusively on a single malware subtype (Transponder spyware), they achieved excellent detection accuracy (99.84%) against 14 entirely different and previously unseen malware subtypes. Their methodology significantly emphasized interpretability through SHAP (Shapley Additive Explanation) and computational efficiency with a minimal feature set which contains only five features, resulting in a lightweight and real-time deployable solution.

Methodology

The methodology for this project begins with working on the related work. It was tested whether LightGBM and XGBoost gradient-boosted learners would achieve better results compared to the random forest algorithm used in the related work.

For this purpose, the same dataset, CIC-MalMem-2022, used in the related work was used. The dataset includes memory dumps of benign files and 3 main types of malware: trojan, spyware and ransomware. These main types are divided into 15 different subtypes. The dataset contains 29298 benign memory dumps and 29298 malware memory dumps. The distribution of subtypes can be seen in the figure below.

Malware type	Malware subtype	Number of instance	Percentage (%)
Trojan Horse	Zeus	1950	3.3
	Emotet	1967	3.4
	Refroso	2000	3.4
	Scar	2000	3.4
	Reconyc	1570	2.7
Spyware	180Sulotions	2000	3.4
	CoolWebSearch	2000	3.4
	Gator	2200	3.8
	Transponder	2410	4.1
	TIBS	1410	2.4
Ransomware	Conti	1988	3.4
	MAZE	1958	3.3
	Pysa	1717	2.9
	Ako	2000	3.4
	Shade	2128	3.6
Total	-	29,298	50.0

Figure 1: Distribution of malware subtype

Based on the improvements made in related work, zero-shot reboot state detection across Android malware families was performed using the dataset CCCS-CIC-AndMal-2020's dynamic analysis part. The dataset contains 53439 rows of 14 malware families. For understanding the behavioral changes of these malware categories and families, six categories of features are extracted after executing the malware in an emulated environment.

- Memory: Metrics of the app's memory consumption (e.g. PssTotal, heap usage, etc.), reflecting how malware utilizes device memory
- API: Invocation of critical Android APIs, which reveal malware behaviors. This includes process management calls (starting/killing processes), executing shell commands, file I/O and database operations, inter-process communications (sending broadcasts, starting services), cryptographic operations, device identifier access, network connection APIs, dynamic code loading (DexClassLoader), SMS sending, etc. Each such API feature indicates that the malware called a particular sensitive function (for example, calls to Runtime.exec for shell commands, IoBridge.open for file access, TelephonyManager.getDeviceId for device IMEI, SmsManager.sendTextMessage, HTTP network requests, etc.).
- Network: Statistics on network usage by the app, such as total bytes/packets sent and received during execution. These features quantify the malware's network behavior (e.g. data exfiltration or C2 communication volume).
- Battery: Indications of battery impact, measured via the number of wakelocks acquired and services used by the app.
- Logcat: The count of log messages generated by the app at various log levels (verbose, debug, info, warning, error). This can indicate how chatty or error-prone the app is during execution.
- Process: The number of processes spawned or interacted with by the malware (e.g. total process count during its execution). This reflects the malware's process-level behavior (such as if it tries to fork new processes).

To collect these runtime features, the malware samples were executed in a controlled analysis environment. The dataset documentation[1] notes that malware was run in an emulated environment with identical system settings. in practice, this was a sandboxed Android virtual machine (emulator) instrumented for monitoring. The research team developed a tool called AndroidAppLyzer[2] for static and dynamic analysis, which includes a custom dynamic analyzer component designed to intercept and log the target API calls and system events of interest. This likely involved instrumenting the Android OS to record when the malware invokes the specified APIs or performs certain actions.

The dynamic analysis platform, the Android emulator, was configured to collect memory stats, logcat output, and OS-reported counters for each app. Because some metrics like battery drain

[1] https://www.unb.ca/cic/datasets/andmal2020.html
[2] https://github.com/ahlashkari/AndroidAppLyzer

cannot be directly measured on an emulator, the analysts relied on related signals, for instance, counting wakelock acquisitions and service usage to infer battery impact. The emulator provided a consistent, isolated environment for every sample, and the instrumentation ensured that events such as file opens, URL connections, or SMS sends by the malware were captured as feature data.

The dataset's creators have not stated an exact duration for dynamic execution on the official webpage. However, a previous CIC Android malware study CIC-AndMal-2017 even defined 3 phases. It used a similar methodology with multiple time windows: they captured behavior for 1–3 minutes immediately after installation, then again up to 15 minutes before a reboot, and 15 minutes after reboot.[3] While the CCCS-CIC-AndMal-2020 analysis may not have waited a full 15 minutes due to the far larger scale, it presumably ran each app for a reasonable interval (often researchers use ~2–5 minutes per run as a practical compromise) to allow background services, network activity, or triggered events to occur. By doing two runs per app, one normal and one after reboot, the analysts maximized the chance of observing both immediate and delayed malicious behaviors. This protocol was specifically designed to overcome the stealth of advanced malware that might try to lie dormant initially.[4] In summary, each malware was executed long enough to collect its memory, API, network, log, etc. features twice, once in a fresh install scenario and once after a system restart, ensuring even time-triggered or stealthy payloads were captured.[5]

By following these procedures, the CIC researchers ensured that each malware's dynamic profile was captured in a reliable way. Every sample was subjected to the same treatment, making the resulting feature data robust for comparative analysis like training machine learning models to classify malware by family or category. The two phase execution is particularly noteworthy as a protocol that significantly improves coverage of malware behavior.

The distribution of malware states (before and after) and malware category distribution can be seen in the figures below.

[3] https://www.unb.ca/cic/datasets/andmal2017.html#:~:text=1,15%20min%20after%20rebooting%20phones
[4] https://www.unb.ca/cic/datasets/andmal2017.html#:~:text=In%20order%20to%20acquire%20a,stealthiness%20of%20an%20advanced%20malware
[5] https://www.techscience.com/csse/v48n5/57946/html#:~:text=has%20two%20instances%20in%20the,6

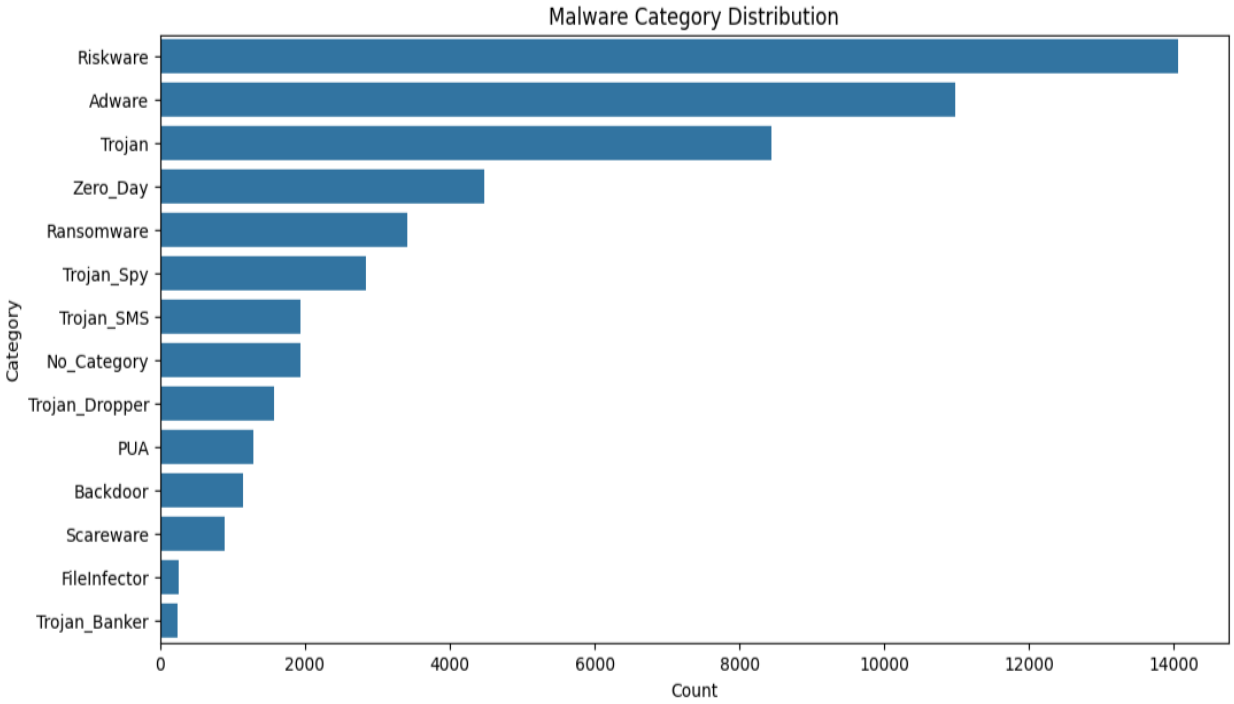


Figure 2: Distribution of Malware categories

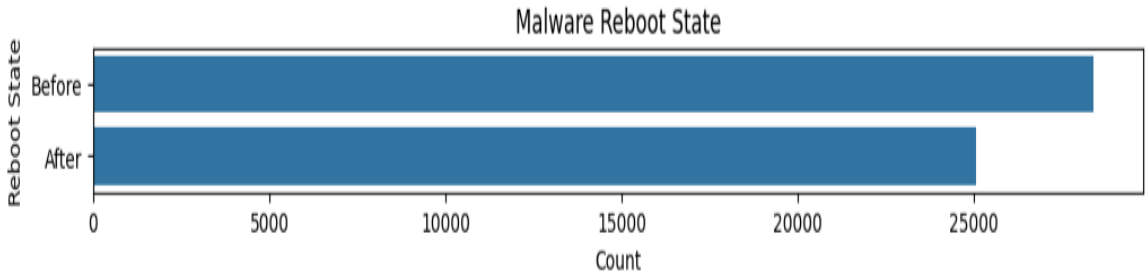


Figure 3: Distribution of Malware States

To prevent unequal distribution of before and after states in the dataset, SMOTE used.

First, binary classification was performed using LightGBM, XGBoost, Random Forest, Logistic Regression and Decision Tree to see which algorithm would give better results. Based on the results obtained, it was decided to use XGBoost in the study.

Afterwards, in order to find the most effective reboot-predictive features, the 5 most important features were found in all malware categories one by one and their total frequency was examined.

Finally, the top-5 most important features per category from from feature extraction process collected and used in zero-shot learning. To enforce zero-shot learning, each model trained on %80

of a single malware category's samples and then tested it on the remaining %20 of that category plus all samples from every other category. The model trained with XGBoost. The results obtained were documented and analyzed.

Results

With the improvements made in the related work, XGBoost and LightGBM outperformed Random Forest in 14 out of 15 malware subtype categories. Only in "Transponder" subtype RF achieved better result. LightGBM achieved the highest accuracy in 10 out of 15 malware subtypes, while XGBoost had the best results in 4 subtypes. LightGBM achieved the highest accuracy, with a score of 99.88% on the Coolwebsearch subtype, narrowly surpassing the highest accuracy reported in the paper %99.84, which was obtained using Random Forest on the Transponder subtype. Comparison across Random Forest, LightGBM and XGBoost can be seen in the figure below.

Malware Subtype	Random Forest (Paper's Work)	XGBoost (My Work)	LightGBM (My Work)
Pysa	0.9974	0.9956	0.9982
Conti	0.9962	0.9973	0.9978
MAZE	0.9953	0.9983	0.9980
Shade	0.9905	0.9986	0.9983
Ako	0.9844	0.9973	0.9980
Transponder	0.9984	0.9980	0.9976
Gator	0.9977	0.9982	0.99880
180Solutions	0.9972	0. 9980	0.9983
TIBS	0.9955	0.9962	0.9959
CoolWebSearch	0.9904	0.9975	0.99884
Reconyc	0.9967	0.9977	0.9956
Emotet	0.9901	0.9969	0.9987
Refroso	0.9810	0.9966	0.9978
Scar	0.9800	0.9974	0.9983
Zeus	0.9800	0.9973	0.9985

Figure 4: Comparison of Accuracy: RF(Paper) vs. XGBoost and LightGBM

These results motivated me to apply XGBoost and LightGBM in my work inspired by this paper.

The best result in binary classification for zero-shot reboot state detection across Android malware families was obtained with XGBoost. Confusion matrices of RF, LightGBM and XGBoost can be seen in figures below.

```
=== Random Forest ===
Time: 5.85s
              precision    recall  f1-score   support

      Before       1.00      0.99      0.99      5676
       After       0.99      1.00      0.99      5676

    accuracy                           0.99     11352
   macro avg       0.99      0.99      0.99     11352
weighted avg       0.99      0.99      0.99     11352
```

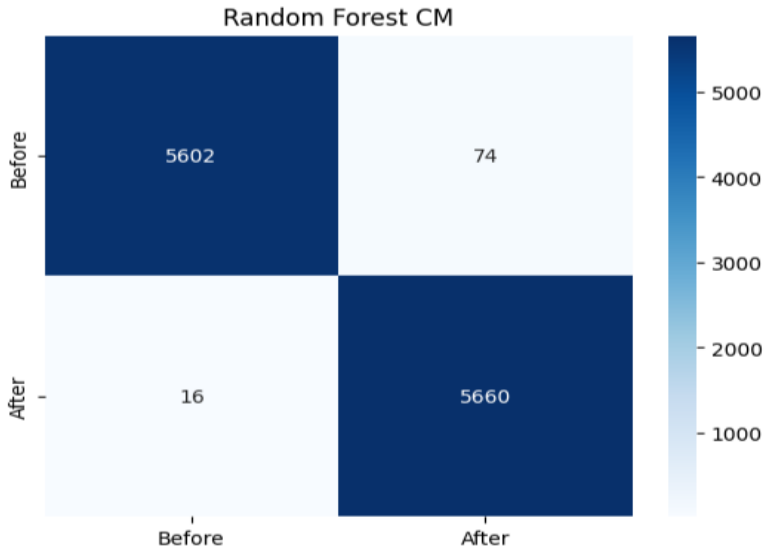


Figure 5: Binary classification results of RF

```
=== LightGBM ===
Time: 2.46s
              precision    recall  f1-score   support

      Before       1.00      1.00      1.00      5676
       After       1.00      1.00      1.00      5676

    accuracy                           1.00     11352
   macro avg       1.00      1.00      1.00     11352
weighted avg       1.00      1.00      1.00     11352
```

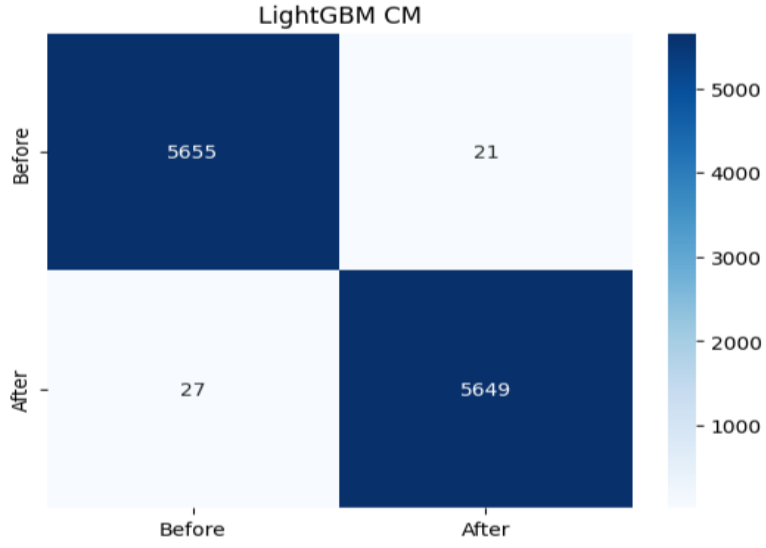


Figure 6: Binary classification results of LightGBM

```
=== XGBoost ===
Time: 2.10s
              precision    recall  f1-score   support

      Before       1.00      1.00      1.00      5676
       After       1.00      1.00      1.00      5676

    accuracy                           1.00     11352
   macro avg       1.00      1.00      1.00     11352
weighted avg       1.00      1.00      1.00     11352
```

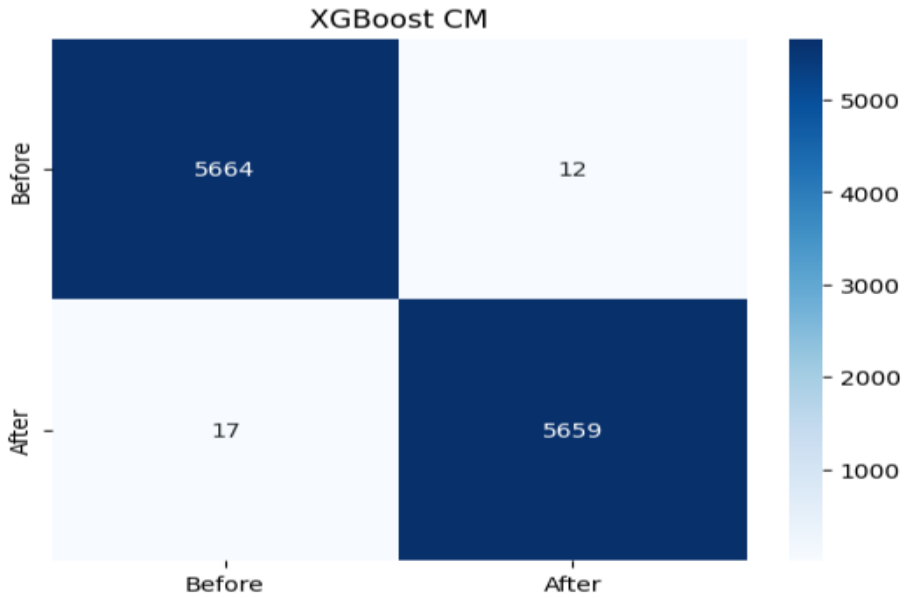


Figure 7: Binary classification results of XGBoost

The features with the highest frequency in all categories in feature extraction using XGBoost can be seen in the figure below.

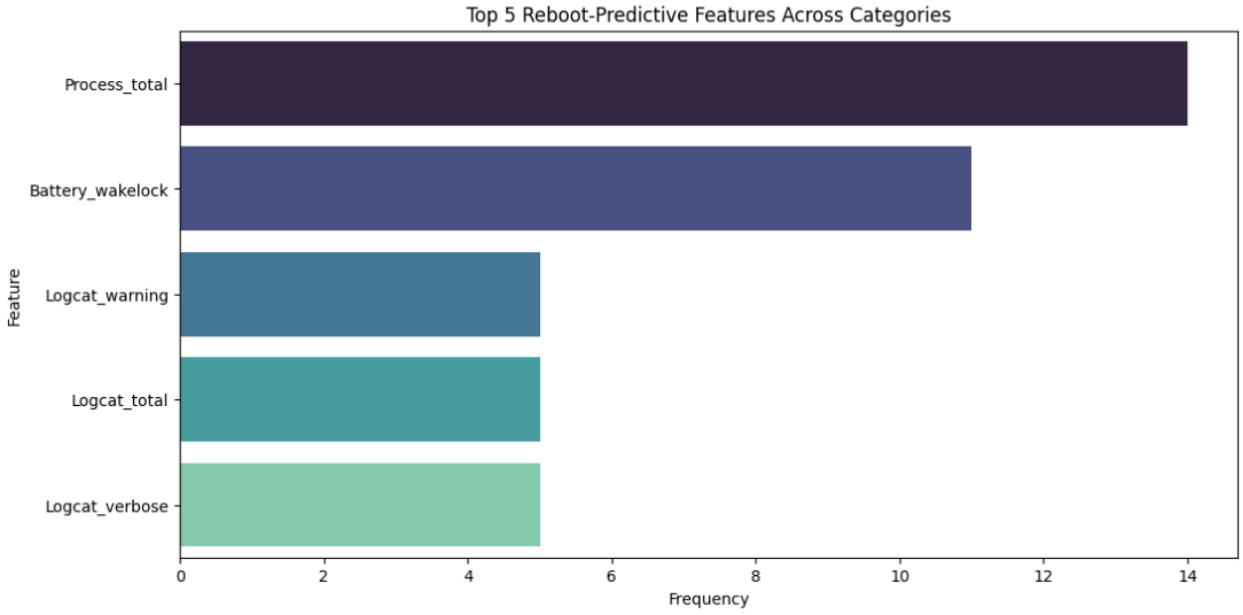
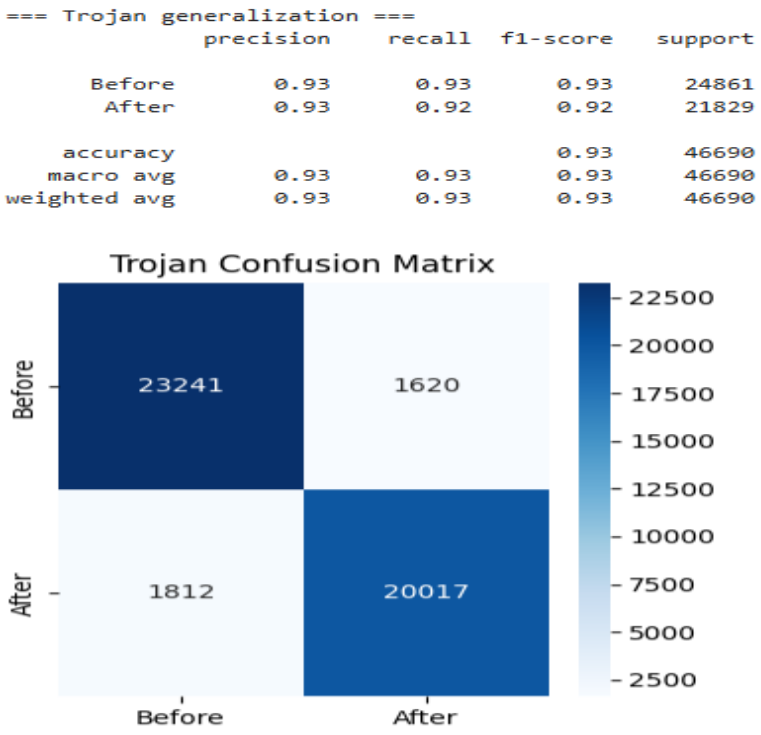


Figure 8: Most effective reboot-predictive 5 features

According to the results obtained at the end of the training using the 5 most important features of each category and using XGBoost, the best result was obtained by Trojan malware with 92.64%. Statistics and confusion matrix of the Trojan malware subtype's can be seen in the figure below.



=== Trojan generalization ===

	precision	recall	f1-score	support
Before	0.93	0.93	0.93	24861
After	0.93	0.92	0.92	21829
accuracy			0.93	46690
macro avg	0.93	0.93	0.93	46690
weighted avg	0.93	0.93	0.93	46690

Figure 9: Trojan Trained Model statistics and confusion matrix

Zero-shot reboot accuracy of each category can be seen in the figure below.

Malware	Before	After
Adware	0.928653	0.886744
Backdoor	0.815525	0.852761
FileInfector	0.757196	0.784481
No_Category	0.921206	0.835586
PUA	0.866363	0.846790
Ransomware	0.878235	0.903674
Riskware	0.772430	0.904507
Scareware	0.909007	0.678534
Trojan	0.934838	0.916991
Trojan_Banker	0.855906	0.795112
Trojan_Dropper	0.947803	0.866645
Trojan_SMS	0.920061	0.770901
Trojan_Spy	0.860133	0.799794
Zero_Day	0.917267	0.893573

Figure 10: Zero-shot reboot accuracy by malware type

SHAP analysis of top-5 features of Trojan-trained model can be seen in the figure below.

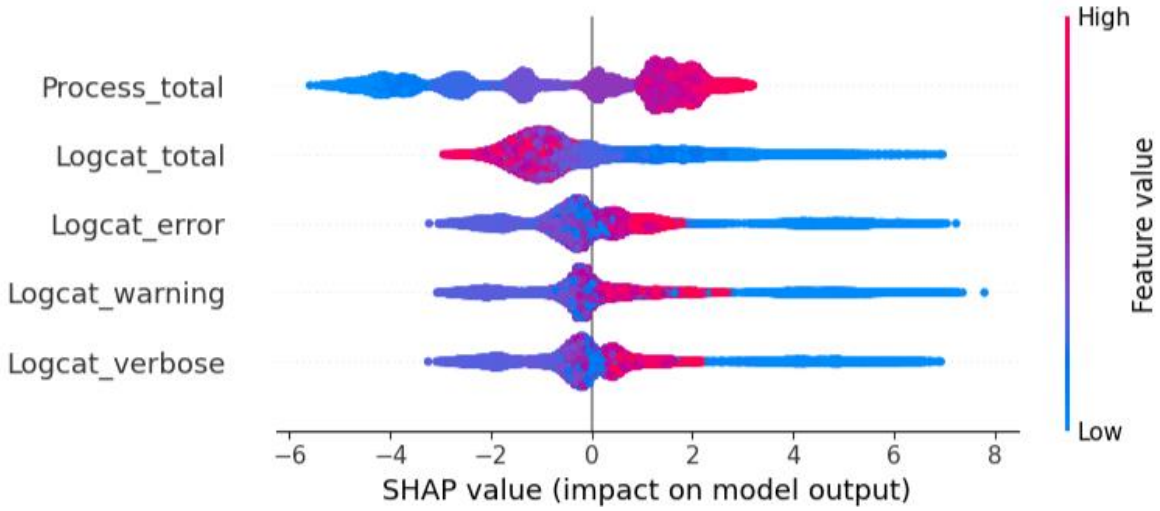


Figure 11: SHAP analysis of top-5 features of Trojan-trained model

SHAP analysis confirmed that total process count and logcat activity drive the model's reboot-state decisions, building trust in predictions.

The repository of this project can be accessed at the following address:
https://github.com/atahanturk/CMP656/

Conclusion

The results obtained in related work were improved using XGBoost and LightGBM, and it was found that using gradient boosted learners gave better results.

Demonstrated that a model trained on just one Android family (e.g. Trojans) can predict pre- vs post-execution with high accuracy (92.64 %) on 13 completely unseen families.

Discovered five dynamic metrics that generalize across all malware categories.

SHAP analysis confirmed that higher process activity and log output reliably indicate "After" execution, making the detection interpretable for security analysts.

References

Oladipo A. Madamidola, Felix Ngobigha, Adnane Ez-zizi (2025). Detecting new obfuscated malware variants: A lightweight and interpretable machine learning approach. Intelligent Systems with Applications, 25, 200472

CIC-MalMem-2022

CCCS-CIC-AndMal-2020

https://www.unb.ca/cic/datasets/andmal2017.html

https://www.techscience.com/csse/v48n5/57946/html#:~:text=has%20two%20instances%20in%20the,6