International Journal on Perceptive and Cognitive Computing (IJPCC) Vol 11, Issue 1 (2025)
https://doi.org/10.31436/ijpcc.v11i1.531

Perceptive Computing for Android Threats: Unveiling Jekyll and Hyde Syndrome in Scareware

Andi Fitriah Abdul Kadir[1], Hairul Nizam Balalo @ Bolalan[2]
[1]Department of Computer Science, International Islamic University Malaysia, 53100, Kuala Lumpur, Malaysia
[2]Commercial Crime Investigation Department (CCID), Level 27, Digital Forensic Investigation Section, Royal Malaysia Police, Menara KPJ, Jalan Tun Razak, 50400 Kuala Lumpur, Malaysia

*Corresponding author andifitriah@iium.edu.my
(Received: 30th December, 2024; Accepted: 10th January, 2024; Published on-line: 30th January, 2025)

Abstract— This paper spotlights Android scareware, relating its deceptive behavior to the dual personality syndrome of Jekyll and Hyde, as described in *The Strange Case of Dr. Jekyll and Mr. Hyde*. Modern scareware employs sophisticated evasion techniques, including metamorphic and polymorphic obfuscation, enabling it to alter its code structure during propagation. Additionally, anti-emulator techniques allow scareware to detect emulation environments and conceal malicious activities. To address these challenges, we propose a hybrid approach that combines static and dynamic analysis, leveraging features derived from unreferenced strings and network flow. This method enhances detection by uncovering scareware's dual behaviors. Using five classifiers, we construct models to address three detection scenarios: identifying malicious Android apps, categorizing apps by scareware type, and classifying apps into scareware families. Tested on a dataset of 1,350 samples, the proposed method outperforms existing approaches, achieving over 90% accuracy across all scenarios with an average false positive rate of just 0.04.

Keywords— Android, dynamic, scareware, static analysis, malware analysis, machine learning

I. Introduction

Recently, scareware has ultimately become an effective attack method for cybercriminals in getting funds; the cybercriminals make money from the malicious application (app) by threatening victims to download the apps or convincing them to pay some amount of money for the fake service. Scareware is a malicious software that poses as legitimate application and falsely claims to detect a variety of threats on the affected mobile devices (i.e., battery issues, files corrupted, account hacked). In scareware attacks, the actual target is the human where it aims to exploit human emotion, which can cause panic, shock, anxiety, or the perception of a threat in order to persuade users into purchasing the app (malware) [1]. In fact, the behavior of Android scareware is like the Jekyll and Hyde syndrome [2], exhibiting dual personalities or behaviors. According to malware reports [4, 5], Android scareware was able to bypass the detection system on the app market, i.e., Google Bouncer, for three consecutive years since 2014. Before Google removed the app from the store, Android scareware with a fake antivirus app named *Virus Shield* had been downloaded 30,000 times from Google Play at a price of $3.99 in 2014. A similar incident occurred in 2015, where another malware variant, *AntiVirus for Android*, was downloaded over a million times at $4.99.

In 2016, *Android.Spy.277.origin* was hidden in more than 100 applications on Google Play, infecting 2.8 million Android devices. Furthermore, another variant, *Street Stick Battle*, saw between one million and five million downloads. More recently, between 2020 and 2024, malware attacks have become increasingly sophisticated. For instance, in 2020, *Joker malware* managed to infiltrate hundreds of apps, compromising millions of devices worldwide. In 2022, *Facestealer* spyware emerged, stealing sensitive user data through fake social media apps, while *SharkBot* in 2023 targeted banking credentials via malicious apps disguised as legitimate financial tools. By 2024, the surge of polymorphic malware such as *Xenomorph* demonstrated advanced evasion techniques, highlighting the persistent threat to Android ecosystems [3].

In the past decade, most of the studies have focused on detecting the mobile malware. According to the survey of securing Android devices [6], the researchers have proposed various techniques such as via app-hardening systems (AppInk), through entrusted app or app-market analysis (RiskRanker, SCanDroid, FlowDroid, DroidScope, DroidRanger, Pegasus, DNADroid, DroidMOSS, Stowaway, ComDroid, ContentScope), by continuous runtime monitoring (TaintDroid, BayesDroid, MockDroid, Apex, FlaskDroid, SEAndroid, Porscha), and based on the install-time checking (Kirin, Pyandrazzi).

However, these studies have focused on the binary detection (malware or non-malware) and not specifically detecting scareware. This general detection mechanism is not enough in detecting the sophisticated malware with metamorphic and polymorphic. The fact that scareware appear as a legitimate program (i.e. contain an actual AV) makes it more difficult for the current detection system to identify it as malicious. Although many studies on Android malware [7, 9, 10, 11] are being actively developed recently, research efforts focused on Android scareware are still inadequate. This is due to the lack of understanding of mobile scareware. Without understanding the behavior of scareware, the detection systems are not capable of providing an accurate recognition of an advanced mobile scareware. A simple illustration to that is the detection rate of current AVs and malware detection tools towards scareware.

A quick scan of one of scareware samples called Fake AV with a ***Md5sum: e46a87522cdb53248c8805880d7a6108*** by VirusTotal [8] shows the poor performance of AVs in detecting scareware. VirusTotal is a web-based service that aggregates over 70 antivirus products and online scan engines for analyzing suspicious malware. However, only about 40 products (63% of detection rate) are able to detect this scareware sample. Particularly, there are five submissions available for this sample (scanned history from January 2016 until April 2018). In addition, we re-uploaded this sample to VirusTotal in December 2024. However, the results show no insignificant different; about 63% of samples are detected by AVs in 2016, 66% in 2017, and 48% in 2024. Even after three consecutive years, the detection rate of the AV products for this particular sample has not improved. The results indicate that the current AVs have some limitations in detecting scareware.

To further evaluate the performance of AV products, we scanned 150 samples of scareware that we have collected from multiple sources [12, 13, 8] with three popular AV products named AVG, Avast, Bitdefender. About 15% of the samples are not detected or in other words seen as legitimate apps by AVG and Avast; and about 12% are not detected by Bitdefender as shown in **Table I**. We labelled these samples as Undetected. The result shows a low detection rate with only 1.3%. Out of 150 scareware samples, HelDroid is only able to detect two samples as scareware. The low performance of the current detection systems has led us to perform an in-depth exploration of scareware. Our research aims to tackle this problem by focusing on the hybrid approach of malware detection, which employs both the static and dynamic analysis methods in order to increase the accuracy of detection. A hybrid approach, specifically utilizing features derived from string analysis and network flow patterns, can effectively detect behavior in scareware.

TABLE I
EXAMPLE OF SCAREWARE SAMPLES DETECTED BY AV

AV	AV Detection	Total	Detection Rate (%)
AVG	AVG#Android/G3P.GP.390306FD2DEC#20160621#16.0.0.4604	35	85
	AVG#Android/G2P.B.DAE4F9C28F18#20160816#16.0.0.4647	24	
	AVG#Android/G2M.W.895675B6B0C1#20160807#16.0.0.4627	20	
	Others	48	
	Undetected	23	
Avast	Avast#Android:Penetho-AA [PUP]#20160621#8.0.1489.320	43	85
	Avast#Android:Provar-A [Trj]#20160816#8.0.1489.320	24	
	Avast#Android:FakePlayer-D[Trj]#20160811#8.0.1489.320	22	
	Others	38	
	Undetected	23	
Bit Defender	BitDefender#Android.Hacktool.Pentr.B#20161223#7.2	44	88
	BitDefender#Android.Trojan.FakeInst.AX#20160807#7.2	39	
	BitDefender#Android.Trojan.AVPass.B#20160621#7.2	24	
	Others	25	
	Undetected	18	

II. BACKGROUND AND RELATED WORK

Analyzing an Android app can be performed in two ways: static or dynamic. Static analysis refers to any techniques that are performed without executing the apps, neither on real devices, nor in emulators or sandboxes. Thus, static analysis can be performed faster than dynamic analysis as the latter requires an appropriate execution environment (i.e. runtime while the apps are executed) in order to extract the behavior.

Static Analysis. This approach can be performed by disassembling its source code without execution where several features are collected from the application itself such as the code executables (string, bytecode, opcode, API) and manifest file properties (permission, intent filter, device and application components). This approach consists of two types:

1. Signature-based: this method is commonly used by anti-virus products where it extracts the semantic patterns and defines a unique signature of malware. Although this detection method is very efficient for known malware, it cannot detect the unknown malware types and the obfuscated and advanced malware. Most of the malware remain undetected because of the limited signature database.

2. Dalvik Bytecode: this method helps in analyzing the app's behavior. e.g., Control and data flow analysis detect the dangerous functionalities performed by malicious apps. Android apps are developed in java language, compiled in java bytecode and then translated to Dalvik byte code. In Android, Dalvik is a register-based VM that interprets the Dalvik Executable (DEX) byte code format.

Dynamic Analysis. In contrast with the static analysis, the dynamic analysis aims to find any malicious behavior of Android app while it is running on any platforms including emulator, sandbox, and smartphone. There are four methods of dynamic analysis:

1. Anomaly-based: this method relies on machine learning algorithms in detecting the malicious behaviors of Android apps. In this case, features that are extracted from known malware are used to train the model for predicting an unknown malware.
2. Taint Analysis: this method typically used for data flow analysis and leakage detection, where it automatically labels the data, keeps track of the data, and records the label of the data.
3. Emulation-based: this method executes the apps in sandbox, where it typically uses Monkey tool to analyze the malicious behavior of app.
4. On-device: this method runs apps on any devices such as computers, smartphones, and tablets.

Most studies that employ static analysis have focused on the manifest file properties (AndroidManifest.xml), which hold the application's metadata. For instance, the application permissions contained in AndroidManifest.xml have been explored by several studies, including DroidRanger [15] and Drebin [14]. In addition, static features extracted from code executables often require additional pre-processing and are commonly used in studies in the form of n-grams, including DroidMOSS [16] and DroidKin [17]. In contrast to our approach, we chose to leverage string as our static feature. This is due to the hyphothesis by Richard et al. [18], where they revealed that the unreferenced strings typically carry hidden information embedded in Android apps. This is proven by the example of GoldDream Trojan app analyzed in their work; GoldDream uploaded stolen information to a remote server with the URL of lebar.gicp.net, this URL became visible only through analysis of unreferenced strings. They evaluated their framework on more than 5,000 apps from 14 different malware families and were able to classify samples with over 99% accuracy. Like the desktop malware, the network traffic is one of the dynamic features that is useful for detecting Android malware. However, due to the lack of a large-scale malware repository and a systematic analysis of network traffic features, the existing research mostly focuses on static analysis. For that reason, we used an automated dynamic analyzer for analyzing malware through network traffic analysis. Instead of using an emulator, we run our samples on smartphones to cope with the advanced malware evasion technique. **Table II** depicts the limitation of static and dynamic analysis from the previous works. The sandboxing term was first introduced by Wahbe et al. [19] in 1993 but in a different context, .i.e., software-based fault isolation. Later, Goldberg et al. in 1996 [20] used the term sandboxing to describe the concept of confining a helper application to a restricted virtual environment for security purposes. Today, a sandbox is often used as a security mechanism for separating running programs, which is to execute suspicious or unverified programs, applications, or codes that may contain malicious code typically from third parties, users or websites, without risking the host machine or OS. There are many sandboxes that have been developed for Android applications; however, the publicly available free sandboxes only offer basic information. Furthermore, some of the authors presented their proposed sandbox but unfortunately, they did not release the sandbox for public use. This includes the Mobile Apps Assessment and Analysis System or known as MAS [21] and DroidInjector [24], a process injection-based dynamic tracking system for runtime behaviors of Android applications.

Research Gap Summary. The primary research gap identified in earlier studies is the lack of a focused approach to detecting scareware specifically, as most existing methods target binary malware detection and fail to account for advanced evasion techniques like metamorphism and polymorphism. Recent advancements in hybrid malware detection emphasize the integration of static and dynamic analysis to enhance accuracy and resilience against obfuscation techniques. However, these methods largely remain limited to broader malware categories without delving into scareware's unique behavioral traits. The proposed study addresses this gap by combining unreferenced string analysis (static) with network flow patterns (dynamic) to effectively detect the dual personality of scareware, similar to the Jekyll and Hyde syndrome. This integration ensures robust detection of both overt and covert malicious behaviors, advancing beyond the capabilities of existing frameworks. By leveraging features that expose scareware-specific characteristics, the study provides a targeted solution previously overlooked in hybrid detection research. Moreover, the adoption of anti-emulation techniques ensures its applicability against modern malware variants designed to evade conventional detection environments.

TABLE II
LIMITATION OF STATIC AND DYNAMIC ANALYSIS

Type	Method	Limitation	Related Work
Static	Signature-based	Cannot detect unknown malware types	[26, 27, 28, 29, 30, 31, 32]
	Dalvik bytecode	More power and memory consumption	[15, 16, 19, 33, 34]
Dynamic	Anomaly-based	Unreliable if a benign app shows same behaviors and can consume more battery and memory	[22, 35, 37, 38]
	Taint analysis	Not suitable for a real-time analysis as its slowdown system	[23, 39, 40, 41]
	Emulation-based	More resource consumption	[25, 42, 43, 44, 45]
	On-device	More power and memory consumption	[35, 36, 46, 47, 48]

III. METHOD AND IMPLEMENTATION

With the rapid growth of malware samples, there exist many solutions that can be used by malware analysts for correlating the different signs of the malicious behavior. However, the output of thesesolutions is focusing more on the malware binary detection (or family) and not directly applicable for the malware type categorization. Therefore, we extend the framework outlined in [49], originally employed for detecting Android financial malware, to identify scareware. This framework facilitates malware detection, encompassing a three-tier detection approach: identifying malicious Android applications, classifying Android apps based on scareware categories, and characterizing Android apps according to scareware families. For the development of the study, a detailed illustration of our proposed conceptual framework is outlined in **Figure 1**. It depicts the major facets of this study, as follows:

1. Collector: responsible for collecting the input data of Android scareware from various sources.
2. Filter: acts as a screening filter to check the similarity of the collected data and to correlate the data with the external sources and the proposed taxonomy.
3. Analytics engine: a hybrid of static and dynamic modules, which involves a process of correlating data from malware string and network flow.
4. Detector: consists of three levels of detection; Level 1: malware binary detection; Level 2: malware category classification; Level 3: malware families characterization, which is to label scareware families.

Dataset. The dataset comprises 1,350 Android apps, including 1,200 benign apps sourced from the Google Play Store based on popularity and 150 scareware samples obtained from VirusTotal [8], security blog [13], and other academic datasets [12, 17, 16] as listed in **Table III**. Initially, over 3,000 APK files were collected from these diverse sources to ensure coverage of various scareware types, temporal variations, geographical distributions, and behavioral patterns.

However, several limitations reduced the final dataset size: 1) **Sample errors:** many collected samples were of poor quality, including issues like Dex errors, unsigned apps, or corrupted files. Unsigned apps could not be installed on emulators or real devices, significantly limiting their usability; 2) **Inconsistent malware labeling:** variations in malware naming conventions across academic and industry sources caused confusion and required time-intensive reorganization. Malware labels were aligned by comparing naming conventions from several antivirus vendors and adopting the majority consensus for each family. Benign apps were selected from the Google Play Store based on popularity, while suspicious apps flagged by more than two antivirus engines on VirusTotal were excluded from the benign set. This labeling approach ensured a reliable benchmark for analysis. The dataset's diversity enhances its generalizability, allowing the detection model to adapt to both older and emerging scareware variants, perform consistently across regions, and recognize a wide range of malicious behaviors. This variety minimizes the risk of overfitting and ensures robust performance against novel or underrepresented threats in real-world scenarios.

TABLE III
THE BREAKDOWN OF ANDROID SCAREWARE BY FAMILY

Year	Malware Family	Number of samples collected	Number of samples analyzed
2011	FakePlayer	150	0
2012	Penetho	150	20
2013	AV Pass	150	20
2013	FakeAV	150	22
2013	FakeFlash	150	6
2013	FakeJobeOffer	9	9
2013	Android Defender	150	17
2013	FakeTaoBao	150	0
2013	Tapsnake	150	9
2014	Virus Shield	150	10
2015	AV for Android	83	10
2015	FakeApp	150	10
2015	FakeApp. AL	150	11
2016	Android Spy.277	9	6
Total		**1751**	**150**

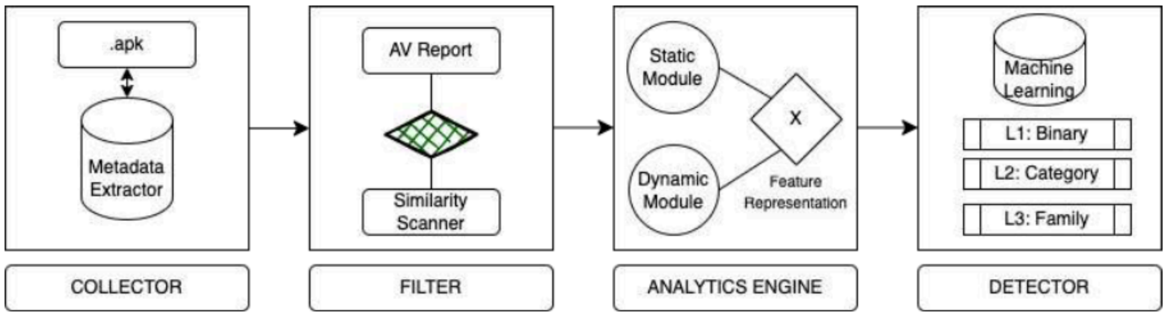

Fig. 1 Overview of the proposed Android scareware detection system

System Configuration. In this section, we discuss the various tools and techniques used for the implementation of the proposed framework. We implemented our framework by using UNIX Shell scripting, Python, and Java programming language. The static analysis implementation of the String Analyzer employs Apktool to decode APKs and Natural Language Toolkit (NLTK) to extract string literal features. In this analysis, we used various Python modules available in the scikit-learn library. Scikit-learn is a Python-based, open source library for handling various data mining and analysis tasks [50]. It provides implementations of a wide range of machine learning algorithms and functionality to generate feature vectors from string n-grams extracted from every APK under the analysis. On the other hand, we used the Shell scripting and Java programming language in the dynamic analysis implementation. The network analyzer app's installation used Shell scripting to run the samples automatically. Machine learning tasks including preprocessing, feature selection, training and testing phases are carried out through Scikit-learn libraries and Weka data mining tool.

Learning Parameters. We employed five common machine learning classifiers including k-Nearest Neighbors (kNN), Support Vector Machine (SVM), Logistic Regression (LR), Naive Bayes (NB), and Random Forest (RF). These algorithms should be calibrated with parameters that ensure maximum performance, which in our case means the maximum classification accuracy and the minimum number of false positives. To avoid over-tuning, we only use 20% of the dataset for parameter tuning. We perform different rounds of experiments to decide on the optimal value for each parameter. Below is the list of parameters that need to be optimized for each classifier.

a) RF: number of trees (set to 100 trees) and minimum number of instances per leaf (set to 1 leaf)
b) NB: type of estimator (use kernel density estimator)
c) kNN: number of neighbors, k (set k to 2 neighbors)
d) SVM: kernel function (use Radial basis function)

Evaluation Metrics. In this section, we explain how the system performance is evaluated. We also discuss various measures used for evaluating the system such as accuracy, F1 measure, Receiver Operating Characteristic (ROC) curve, and false positive rate. The goal of the classification model is to correctly classify an input sample to one of the output classes from a set of discrete output categories. Our Android malware detection framework trains a classifier over training samples. The classifier then predicts the category of APKs in the test data. The best way to represent such output predictions of the classification model is to use a confusion matrix. A confusion matrix is an N × N contingency table, where N is the number of output labels. It shows the number of samples correctly and incorrectly classified by the model as compared to the actual target output values. For example, consider a binary classification model with output labels as either Positive or Negative.

IV. Experiment and Result

In this section, we present the results obtained in detecting scareware statically and dynamically, as described in Section 4. In our experimental study, we focused on analysis of binary classification, category classification, and family classification. In order to evaluate the detection performance of the proposed systems, we split the data into 60% of train-set and 40% of test-set. We reported three metrics in each scenario:

a) **Accuracy:** refers to the overall classification accuracy measure, which is given by the percentage of correctly classified instances.
b) **F-measure:** considers class imbalance, which represents a weighted average of recall and precision.
c) **FPR** (false positive rate): defines the ratio between the number of negative events incorrectly

categorized as positive (false positives) and the total number of actual negative events.

TABLE IV
SCAREWARE DETECTION RESULTS WITH 3-GRAM WORD OF STRING

Algorithm	Binary Detection (2-classes)			Category Classification (3-classes)			Family Characterization (13-classes)		
	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR
NB	55.41	89.93	0.516	68.733	95.00	0.291	48.31	94.31	0.444
KNN	87.31	98.94	0.113	78.63	95.99	0.194	60.28	95.32	0.262
SVM	90.31	96.46	0.185	82.98	96.57	0.165	69.13	96.55	0.608
LR	91.83	97.14	0.124	84.27	97.01	0.148	71.17	97.17	0.505
RF	91.72	97.38	0.014	86.91	97.27	0.108	72.87	97.12	0.162

TABLE V
SCAREWARE DETECTION RESULT WITH NETWORK FLOW

Algorithm	Binary Detection (2-classes)			Category Classification (3-classes)			Family Characterization (13-classes)		
	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR	F-Measure	Accuracy	FPR
NB	98.40	98.33	0.031	94.20	93.67	0.033	92.40	92.33	0.031
KNN	98.80	98.83	0.075	95.90	95.83	0.077	92.50	92.83	0.076
SVM	85.30	90.00	0.900	85.30	90.00	0.900	85.30	90.00	0.900
LR	99.00	99.00	0.075	95.00	94.67	0.063	90.80	90.17	0.046
RF	99.30	99.30	0.045	96.50	97.17	0.047	92.60	92.83	0.046

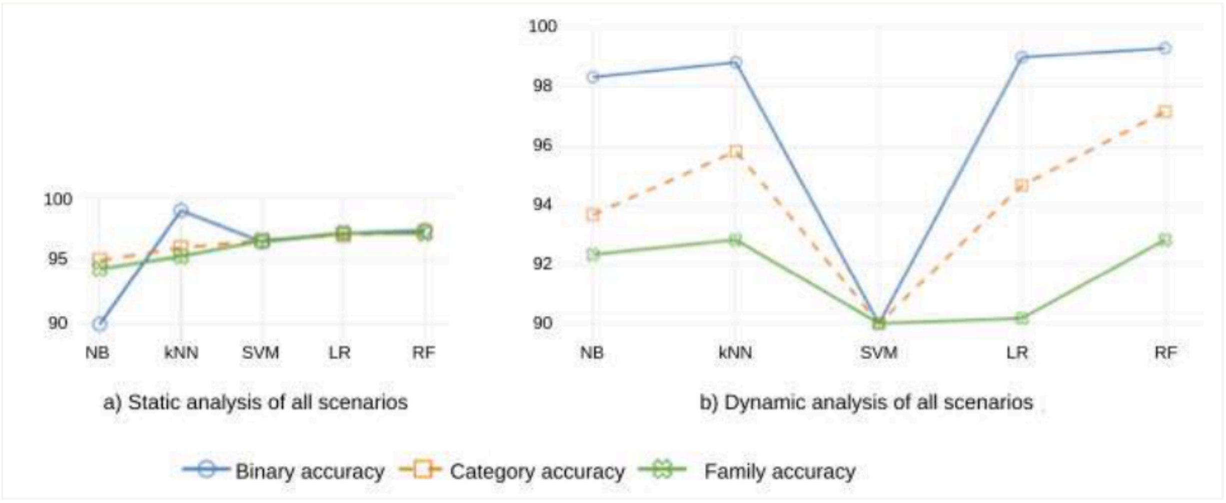

Fig. 2 Accuracy comparison of static and dynamic analysis for all scenario

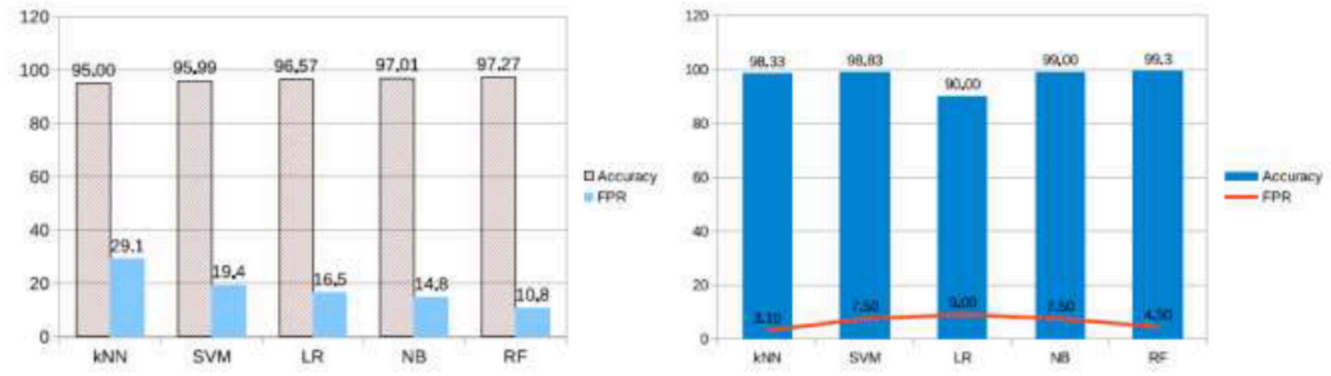

Fig. 3 3-gram scareware detection (left) and netflow scareware detection (right)

For static analysis, we tested up to 3-gram of word token with all classifers. On average, 3-gram yields the highest precision. Thus, in this paper, we reported the result of the 3-gram for all three scenarios (**Table IV**). KNN surpasses RF in the binary detection, but RF performs the best in all scenarios: 97.38% accuracy with binary detection, 97.27% accuracy with category detection, and 97.12% accuracy with family detection with an average of 0.095 FPR.

Similar to the static analysis, RF also performs the best in all scenarios for dynamic analysis: 99.30% accuracy with binary detection, 97.17% accuracy with category detection, and 92.83% accuracy with family detection with an average of 0.046 FPR (**Table V**). Since both static and dynamic results (**Figure 2**) yield a high accuracy and low FPR, there is no need for us to have the integrated feature vectors between the 3-gram unreferenced strings and the 80 nominal network ow features for the purpose of increasing the accuracy. Our results demonstrates that the raw features of each analysis is adequate in detecting scareware, classifying the category, and characterizing its family accurately with very low percentage of FPR. **Figure 3** presents the result of scareware for both static and dynamic where RF also outperformed other classifiers with more than 97% and 99.30% accuracy respectively.

RF outperformed other classifiers due to its ability to handle high-dimensional feature spaces and its robustness against overfitting, especially when analyzing diverse datasets. By leveraging ensemble learning, RF builds multiple decision trees and aggregates their predictions, which enhances accuracy and reduces bias. Compared to simpler classifiers like NB, RF's capacity to capture non-linear relationships between features contributed to its superior performance in detecting the nuanced behaviors of scareware.

The strange case of Android.Spy.277. In order to evaluate our framework, we conducted a case study of one of the sophisticated malware family in our dataset (AndroidSpy). Similar to Jekyll's behavior, AndroidSpy first appeared as legitimate apps that offered services like games, wallpapers, photo editing apps. But, once installed, the app transforms into Hyde and becomes malicious through a backdoor. These apps had been downloaded by almost 3 million users [5]. The attacker can remotely download a malicious APK called ***polacin.io*** to the victim's device. Once infected, the Android device sends a wide array of information about the phone to command and control servers (C&C), including phone IMEI number, email address, sms messages, and location. What's more, AndroidSpy performs and additional malicious act through unwanted advertisements via popups and notification's bar. Victims are induced into installing fraudulent apps via fake warning of battery issues that can be solved by downloading fake utilities. By clicking on these fake alerts, for example, brings victims to the landing pages for Android optimization applications such as Turbo Cleaner, SuperB Cleaner (Boost Clean). This behavior reveals that AndroidSpy have several layered of attacks to sustain their malicious behavior. To further analyzed the stealthiness of this AndroidSpy, we analyzed all 6 samples that we have by using three malware scanners: AndroTotal[1], Joe Sandbox[2], and VirusTotal[3]. The results depict that most of the AV are not able to detect this type of scareware. Out of 100 AV deployed on these scanners, only 25 of them are able to detect AndroidSpy as maliciouss apps (**Table VI**). With our approach, we managed to detect AndroidSpy accurately (detected 5 apps out of 6) with 99.42% of ROC area value (the area under the ROC curve is a measure of how well a

[1] https://andrototal.org/
[2] https://www.joesandbox.com/
[3] https://www.virustotal.com

parameter can distinguish between two groups malicious and benign. The true positive rate surpasses 80% when the false positive rate is greater than 60%. Therefore the closer the ROC curve is to the upper left corner (true positive rate), the higher the overall accuracy of the test.

TABLE VI
COMPARISON RESULTS OF ANDROIDSPY ANALYSIS

SCANNER	AV DETECTED	TOTAL AV
VIRUSTOTAL	20	61
ANDROTOTAL	3	8
JOE SANDBOX	2	31

V. LIMITATION AND DISCUSSION

The study of Android scareware is stimulating. As this is the first study of its type to systematically detecting scareware, researchers not only have a new opportunity for research but also several challenges:

a) **Dataset biases:** the study concluded that integrating static (unreferenced strings) and dynamic (network flow) feature vectors was unnecessary, as both feature sets independently achieved high accuracy and low false positive rates. This simplification reduces computational overhead, making the system more suitable for real-world deployment on resource-constrained devices. However, potential dataset biases such as the reliance on scareware samples predominantly flagged by certain antivirus engines could influence metrics, potentially skewing detection results toward these engines' strengths.

b) **Scareware labeling**: the naming convention for malware labeling is inconsistent across both academic and industry fields, leading to confusion and inefficiencies during reorganization. The inconsistent labeling practices among different antivirus vendors and researchers further complicate the process. For instance, services like VirusTotal aggregate results from multiple antivirus engines, and discrepancies in detection can create ambiguity. Some engines may flag an APK as scareware, while others classify it as benign, requiring manual interpretation to resolve conflicts. To address this, we compare malware labeling from several antivirus vendors and follow the majority consensus based on the most frequent label for a specific malware family. However, this technique is entirely manual, introducing the risk of human error due to subjective judgment, inconsistencies in labeling criteria, or varying levels of expertise. These errors can impact the accuracy of malware labeling, potentially affecting the dataset's reliability and the model's performance.

c) **Scalability**: the proposed system demonstrates promising scalability for large-scale applications due to its reliance on lightweight static and dynamic analysis, avoiding the complexity of integrated feature processing. Its modular design allows seamless scaling across distributed detection infrastructures, where static analysis can filter benign apps rapidly, while dynamic analysis is reserved for suspected cases. However, the real-world deployment would require robust automation for sample collection and labeling to handle the vast influx of applications. Additionally, optimizing computational resources and incorporating cloud-based processing could enhance throughput and maintain low latency, ensuring effective operation in environments like app stores and enterprise security systems.

VI. CONCLUSION AND FUTURE WORK

The syndrome of Jekyll and Hyde is increasingly adopted by Android malware especially scareware. In this research, we presented a novel combination of both static and dynamic analysis based specifically on features derived from the unreferenced string and network flow. The key idea is to be able to detect the Jekyll behavior at early stage through static analysis before transforming into Hyde behavior, which can later be detected through dynamic analysis. We demonstrated that this combination could identify three scenarios: detecting malicious Android apps, classifying Android apps with respect to scareware category, and characterizing Android apps according to scareware family. The experimental results show that the proposed method achieves high accuracy of over 90% for all three scenarios with a very low false positive rate of 0.04 on average.

The proposed framework effectively detects scareware using a hybrid approach that combines static and dynamic analysis, but there are multiple areas for further enhancement and exploration. First, while the dataset used in this study is diverse, consisting of 1,350 apps, it is relatively small compared to the vast number of Android applications available globally. Future work should prioritize expanding the dataset by incorporating more samples, including new and evolving malware families like ransomware, adware, spyware, and phishing apps. This would not only improve the robustness of the detection model but also allow it to adapt to emerging threats. Moreover, creating a publicly available and standardized scareware dataset could contribute significantly to research in this domain and enable consistent benchmarking across studies.

Another critical area is automating the labeling process to address inconsistencies in malware family naming conventions across different antivirus engines and research datasets. Advanced machine learning algorithms, consensus-driven approaches, or natural language processing techniques could be employed to harmonize malware labels more efficiently and reduce manual intervention. Additionally, future research could explore dynamic labeling systems that update in real-time as malware evolves, ensuring that datasets remain relevant and up-to-date.

In terms of scalability, the framework's computational efficiency must be optimized for real-world deployment. While the current approach effectively utilizes both static and dynamic features, further refinement could focus on reducing processing time and resource requirements. Implementing lightweight versions of the framework tailored for use on mobile devices or integrating cloud-based or distributed systems would make the solution more accessible for large-scale applications. For example, app stores or corporate security networks could use such a system to analyze vast numbers of apps without significant delays. The potential integration of federated learning could also be explored, allowing the framework to continuously improve and update its detection models using decentralized data while maintaining user privacy and reducing bandwidth constraints.

Additionally, future studies should investigate other advanced evasion techniques employed by scareware and malware in general, such as anti-debugging, anti-sandboxing, and advanced anti-emulation behaviors. Understanding and addressing these techniques would enhance the framework's ability to detect increasingly sophisticated threats. It would also be valuable to explore the integration of additional data sources, such as user reviews, app metadata, and permissions, to complement the current static and dynamic features and improve the overall detection accuracy.

Finally, extending the framework to address other types of malware beyond scareware could demonstrate its broader applicability. Experiments on categories like financial malware, ransomware, and spyware could provide insights into the model's generalizability and practical effectiveness across diverse threat landscapes. Real-world testing in various environments, such as corporate IT systems, app marketplaces, and government agencies, would further validate its utility and identify potential areas for improvement. Addressing these challenges will ensure the framework remains relevant and adaptable in the rapidly evolving field of cybersecurity, paving the way for a comprehensive solution to combat mobile threats on a global scale.

Acknowledgment

The authors hereby acknowledge the review support offered by the IJPCC reviewers who took their time to study the manuscript and find it acceptable for publishing.

Conflict of Interest

The authors declare that there is no conflict of interest

References

[1] J. Giles, "Scareware: the inside story," *New Scientist*, vol. 205, no. 2753, pp. 38–41, 2010.

[2] R. L. Stevenson, "Strange case of dr jekyll and mr hyde," in *Medicine and Literature, Volume Two*, CRC Press, 2018, pp. 105–118.

[3] Kaspersky, "Polymorphic Malware on Android: The Rise of Xenomorph," 2024. [Online]. Available: https://www.kaspersky.com.

[4] "Ad fraud, scareware slinger android.spy. 277.origin found in more than 100 apps," 2016. [Online]. Available: https://www.theregister.co.uk/2016/04/26.

[5] "Scareware app downloaded over a million times from google play," 2015. [Online]. Available: http://researchcenter.paloaltonetworks.com/2015/01/scareware-appdownloaded-million-times-google-play/.

[6] D. J. Tan, T. W. Chua, and V. L. Thing, "Securing android: a survey, taxonomy, and challenges," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 58, 2015.

[7] A. I. Ali-Gombe, B. Saltaformaggio, D. Xu, and G. G. Richard III, "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming," *Computers & Security*, vol. 73, pp. 235–248, 2018.

[8] Virus Total. [Online]. Available: https://www.virustotal.com/en/.

[9] C. Lyvas, C. Lambrinoudakis, and D. Geneiatakis, "Dypermin: Dynamic permission mining framework for android platform," *Computers & Security*, vol. 77, pp. 472–487, 2018.

[10] Y. Zhuang, "The performance cost of software obfuscation for android applications," *Computers & Security*, vol. 73, pp. 57–72, 2018.

[11] H. Meng, V. L. Thing, Y. Cheng, Z. Dai, and L. Zhang, "A survey of android exploits in the wild," *Computers & Security*, vol. 76, pp. 71–91, 2018.

[12] A. H. Lashkari, A. F. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *Proceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST)*, 2018.

[13] Virus Total, "Contagio mobile malware mini dump," 2016. [Online]. Available: http://contagiominidump.blogspot.ca/.

[14] D. Arp et al., "Drebin: Effective and explainable detection of android malware in your pocket," in *NDSS*, vol. 14, pp. 23–26, 2014.

[15] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off of my market: detecting malicious apps in official and alternative android markets," in *NDSS*, vol. 25, pp. 50–52, 2012.

[16] W. Zhou et al., "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, pp. 317–326, 2012.

[17] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Droidkin: Lightweight detection of android apps similarity," in *International Conference on Security and Privacy in Communication Systems*, pp. 436–453, Springer, 2014.

[18] R. Killam and N. Stakhanova, "Android malware classification through analysis of string literals," in *Analytics for Cybersecurity and Online Safety*, 2016.

[19] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham, "Efficient software-based fault isolation," in *ACM SIGOPS Operating Systems Review*, vol. 27, pp. 203–216, 1994.
[20] I. Goldberg et al., "A secure environment for untrusted helper applications: Confining the wily hacker," in *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, vol. 6, p. 11, 1996.
[21] C. W. Tien, T. Y. Huang, T. C. Huang, W. H. Chung, and S. Y. Kuo, "MAS: Mobile-apps assessment and analysis system," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 145–148, 2017.
[22] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "MADAM: A multi-level anomaly detector for Android malware," in *MMM-ACNS 2012*, vol. 12, pp. 240–253, Springer, 2012.
[23] W. Enck *et al.*, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems*, vol. 32, no. 2, p. 5, 2014.
[24] W. Fan, Y. Sang, D. Zhang, R. Sun, and Y. Liu, "DroidInjector: A process injection-based dynamic tracking system for runtime behaviors of Android applications," *Computers & Security*, vol. 70, pp. 224–237, 2017.
[25] "Android malware toolkit for malware analysis." [Online]. Available: http://dunkelheit.com.br/amat/analysis/index_en.php
[26] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "AndroSimilar: Robust statistical feature signature for Android malware detection," in *Proceedings of the 6th International Conference on Security of Information and Networks*, pp. 152–159, 2013.
[27] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of Android malware through static analysis," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 576–587, 2014.
[28] M. Zheng, M. Sun, and J. C. S. Lui, "Droid analytics: A signature based analytic system to collect, extract, analyze and associate Android malware," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 163–171, 2013.
[29] R. Sato, D. Chiba, and S. Goto, "Detecting Android malware by analyzing manifest files," *Proceedings of the Asia-Pacific Advanced Network*, vol. 36, pp. 17–23, 2013.
[30] C. Y. Huang, Y. T. Tsai, and C. H. Hsu, "Performance evaluation on permission-based detection for Android malware," in *Advances in Intelligent Systems and Applications—Volume 2*, Springer, 2013, pp. 111–120.
[31] B. Sanz *et al.*, "PUMA: Permission usage to detect malware in Android," in *International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions*, Springer, 2013, pp. 289–298.
[32] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka, "Towards formal analysis of the permission-based security model for Android," in *2009 Fifth International Conference on Wireless and Mobile Communications*, pp. 87–92, 2009.
[33] J. Kim, Y. Yoon, K. Yi, and J. Shin, "SCANDAL: Static analyzer for detecting privacy leaks in Android applications," in *Proceedings of the Mobile Security Technologies (MoST)*, 2012.
[34] E. R. Wognsen, H. S. Karlsen, M. C. Olesen, and R. R. Hansen, "Formalisation and analysis of Dalvik bytecode," *Science of Computer Programming*, vol. 92, pp. 25–55, 2014.
[35] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pp. 15–26, 2011.
[36] P. Irolla and E. Filiol, "Glassbox: Dynamic analysis platform for malware Android applications on real devices," *arXiv preprint arXiv:1609.04718*, 2016.
[37] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Andromaly: A behavioral malware detection framework for Android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.
[38] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "AntiMalDroid: An efficient SVM-based malware detection framework for Android," in *International Conference on Information Computing and Applications*, Springer, 2011, pp. 158–166.
[39] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, pp. 1–6, 2014.
[40] G. Sarwar, O. Mehani, R. Boreli, and M. A. Kaafar, "On the effectiveness of dynamic taint analysis for protecting against private information leaks on Android-based devices," in *SECRYPT 2013*, 2013.
[41] N. Andronio, S. Zanero, and F. Maggi, "Heldroid: Dissecting and detecting mobile ransomware," in *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2015, pp. 382–404.
[42] L. K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in *Proceedings of the 21st USENIX Security Symposium*, pp. 569–584, 2012.
[43] T. Bläsing, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak, "An Android application sandbox system for suspicious software detection," in *2010 5th International Conference on Malicious and Unwanted Software*, pp. 55–62, 2010.
[44] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, pp. 424–436, 2018.
[45] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic reconstruction of Android malware behaviors," in *NDSS*, 2015.
[46] S. Mutti *et al.*, "BareDroid: Large-scale analysis of Android apps on real devices," in *Proceedings of the 31st Annual Computer Security Applications Conference*, pp. 71–80, 2015.
[47] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Emulator vs real phone: Android malware detection using machine learning," in *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*, pp. 65–72, 2017.
[48] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *2012 IEEE Symposium on Security and Privacy*, pp. 95–109, 2012.
[49] A. F. Abdul Kadir, *A Detection Framework for Android Financial Malware*. M.S. thesis, University of New Brunswick, 2018.
[50] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011