AndroShow : Pattern Identification of Obfuscated Android Malware Application

By

Md. Omar Faruque Khan Russel

ID : 152-35-1170

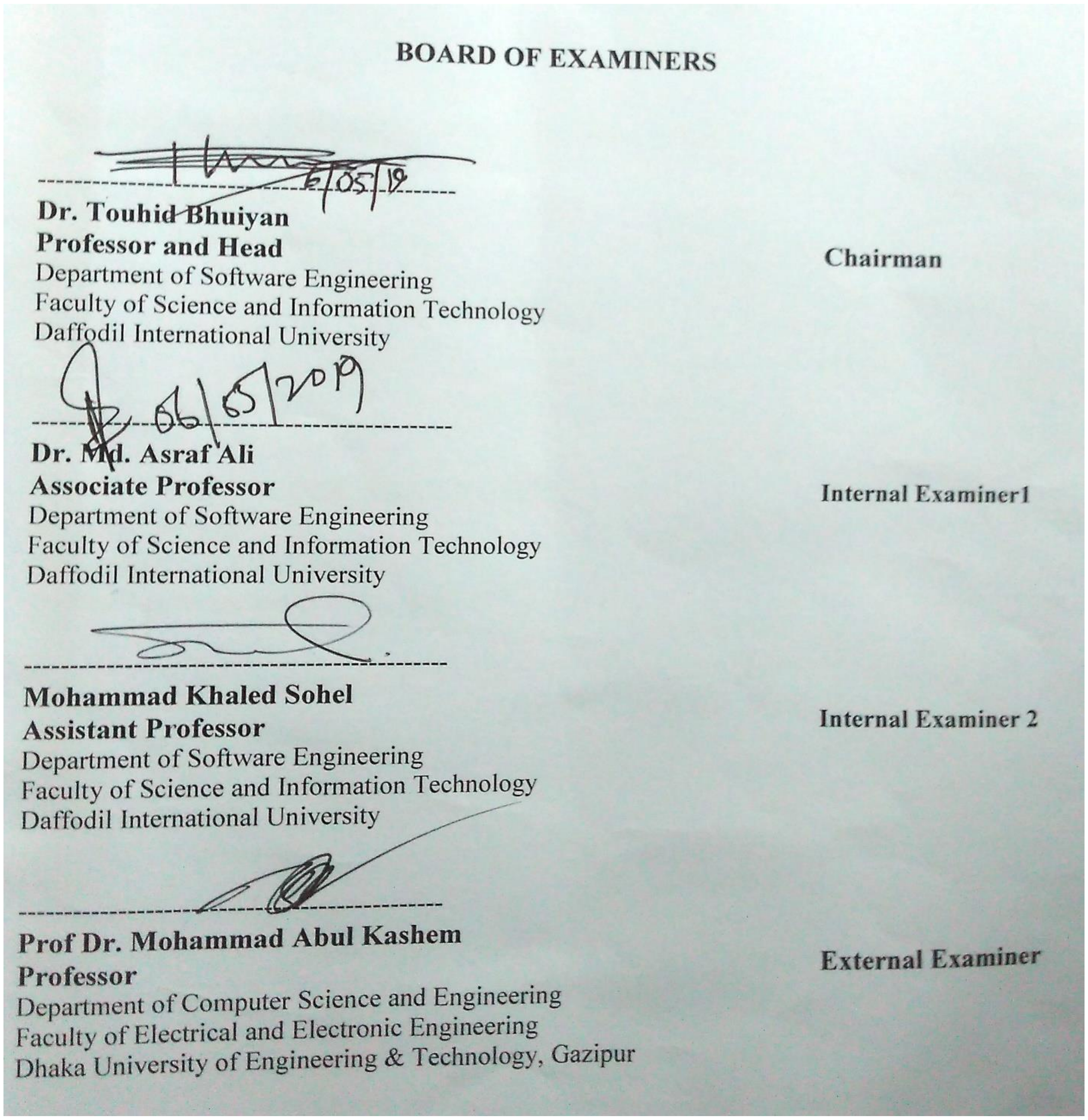
A thesis submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Software Engineering

Department of Software Engineering
DAFFODIL INTERNATIONAL UNIVERSITY

Spring – 2019

APPROVAL

This **Thesis** titled **AndroShow : Pattern Identification of Obfuscated Android Malware Application,** submitted by **MD. OMAR FARUQUE KHAN RUSSEL, ID: 152-35-1170** to the Department of Software Engineering, Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Software Engineering (SWE) and approved as to its style and contents.



BOARD OF EXAMINERS

[illegible] 6/05/19

Dr. Touhid Bhuiyan
Professor and Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Chairman

[illegible] 06/05/2019

Dr. Md. Asraf Ali
Associate Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner1

[illegible]

Mohammad Khaled Sohel
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Internal Examiner 2

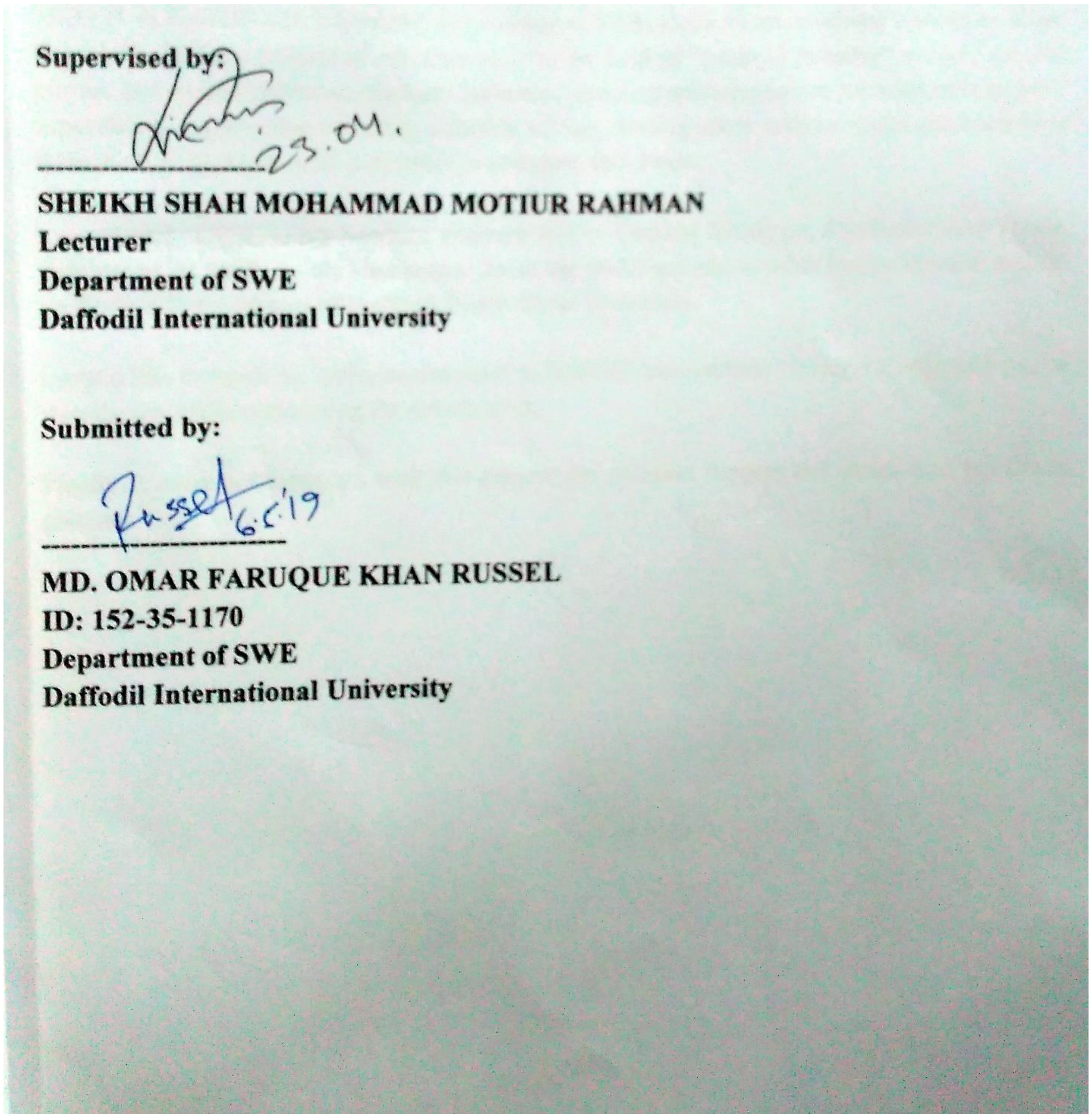
[illegible]

Prof Dr. Mohammad Abul Kashem
Professor
Department of Computer Science and Engineering
Faculty of Electrical and Electronic Engineering
Dhaka University of Engineering & Technology, Gazipur

External Examiner

THESIS DECLARATION

We hereby declare that, this thesis has been done by me under the supervision of **SHEIKH SHAH MOHAMMAD MOTIUR RAHMAN**, **Lecturer**, Department of SWE, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.



Supervised by:

[illegible] 23.04.

SHEIKH SHAH MOHAMMAD MOTIUR RAHMAN
Lecturer
Department of SWE
Daffodil International University

Submitted by:

Russel 6.5.19

MD. OMAR FARUQUE KHAN RUSSEL
ID: 152-35-1170
Department of SWE
Daffodil International University

ACKNOWLEDGEMENT

First, we express our heartiest thanks and gratefulness to almighty Allah for his divine blessing makes us possible to complete the final year thesis successfully.

I really grateful and wish my profound my indebtedness to **SHEIKH SHAH MOHAMMAD MOTIUR RAHMAN**, **Lecturer**, Department of SWE, Daffodil International University. Deep Knowledge & keen interest of my supervisor in the field of "Android Security" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this thesis.

I would like to express my heartiest gratitude to **Dr**. **Touhid Bhuiyan**, **Professor and Head**, Department of SWE, for his kind help to finish my thesis and also to other faculty member and the staff of SWE department of Daffodil International University.

I would like to thank my entire course mate in Daffodil International University, who took part in this discuss while completing the course work.

Finally, I must acknowledge with due respect the constant support and patients of my loving parents.

TABLE OF CONTENTS

APPROVAL **i**

THESIS DECLARATION **i**

ACKNOWLEDGEMENT **iii**

TABLE OF CONTENTS **iv**

LIST OF TABLES **vii**

LIST OF FIGURES **x**

ABSTRACT **xi**

CHAPTER 1: INTRODUCTION **1**

1.1 Background Study 1

1.2 Motivation of the Research 1

1.3 Problem Statement 2

1.4 Research Question 2

1.5 Research Objective 2

1.6 Research Contribution 2

1.7 Thesis Organization 3

CHAPTER 2: LITERATURE REVIEW **4**

2.1 Previous Study 4

2.2 FEATURE ENGINEERING **7**

2.2.1 Permission 7

2.2.2 Intent Filter 8

2.2.3 API Call 9

2.2.4 System Call 10

2.2.5 App Component 11

CHAPTER 3: RESEARCH METHODOLOGY **14**

3.1 Dataset 14

3.2 Feature 14

3.3 Tool 15

3.4 Environment 15

3.5 Procedure 15

CHAPTER 4: RESULT AND DISCUSSION **17**
4.1 PERMISSION ANALYSIS **17**
4.1.1 Trivial Encryption 18
4.1.2 String Encryption 19
4.1.3 Reflection Encryption 20
4.1.4 Class Encryption 21
4.1.5 Combination of Trivial and String Encryption 22
4.1.6 Combination of Trivial, String and Reflection Encryption 23
4.1.7 Combination of Trivial, String, Reflection and Class Encryption 24
4.2 API CALL ANALYSIS **25**
4.2.1 Trivial Encryption 26
4.2.2 String Encryption 27
4.2.3 Reflection Encryption 28
4.2.4 Class Encryption 29
4.2.5 Combination of Trivial and String Encryption 30
4.2.6 Combination of Trivial, String and Reflection Encryption 31
4.2.7 Combination of Trivial, String, Reflection and Class Encryption 32
4.3 SYSTEM COMMAND ANALYSIS **33**
4.3.1 Trivial Encryption 33
4.3.2 String Encryption 34
4.2.3 Reflection Encryption 35
4.3.4 Class Encryption 36
4.3.5 Combination of Trivial and String Encryption 37
4.3.6 Combination of Trivial, String and Reflection Encryption 38
4.3.7 Combination of Trivial, String, Reflection and Class Encryption 39
4.4 INTENT FILTER ANALYSIS **40**
4.4.1 Trivial Encryption 41
4.4.2 String Encryption 42
4.4.3 Reflection Encryption 43
4.4.4 Class Encryption 44
4.4.5 Combination of Trivial and String Encryption 45

4.4.6 Combination of Trivial, String and Reflection Encryption 46
4.4.7 Combination of Trivial, String, Reflection and Class Encryption 47
4.5 APP COMPONENT ANALYSIS 48
4.5.1 Trivial Encryption 48
4.5.2 String Encryption 49
4.5.3 Reflection Encryption 50
4.5.4 Class Encryption 51
4.5.5 Combination of Trivial and String Encryption 52
4.5.6 Combination of Trivial, String and Reflection Encryption 53
4.5.7 Combination of Trivial, String, Reflection and Class Encryption 54
CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS 56
5.1 Findings and Contributions 56
5.2 Recommendations for Future Works 56
REFERENCES 56

LIST OF TABLES

Table 2.1: Some Previous Works on Permission 8
Table 2.2: Existing Researches on Intent Filter 9
Table 2.3: Existing Researches on API Call 10
Table 2.4: Existing Researches on System Call 11
Table 2.5: Existing Researches on App Component 13
Table 4.1: Short Version of Features 17
Table 4.2: Shortened Name for Permissions 17
Table 4.3: A Permission Pattern of Trivial Enc. 18
Table 4.4: Family Permission Pattern of Trivial Enc. 18
Table 4.5: A Permission Pattern of String Enc. 19
Table 4.6: Family Permission Pattern of String Enc. 20
Table 4.7: A Permission Pattern of Reflection Enc. 21
Table 4.8: Family Permission Pattern of Reflection Enc. 21
Table 4.9: A Permission Pattern of Class Enc. 22
Table 4.10: Family Permission Pattern of Class Enc. 22
Table 4.11: A Permission Pattern of Trivial+String Enc. 23
Table 4.12: Family Permission Pattern of Trivial+String Enc. 23
Table 4.13: A Permission Pattern of Trivial+String+Reflection Enc. 24
Table 4.14: Family Permission Pattern of Trivial+String+Reflection Enc. 24
Table 4.15: A Permission Pattern of Trivial+String+Reflection+Class Enc. 25
Table 4.16: A Family Permission Pattern of Trivial+String+Reflection+Class Enc. 25
Table 4.17: Shortened Name for API Call 25
Table 4.18: An API Call Pattern of Trivial Enc. 26
Table 4.19: Family API Call Pattern of Trivial Enc. 26
Table 4.20: An API Call Pattern of String Enc. 27
Table 4.21: Family API Call Pattern of String Enc. 27
Table 4.22: An API Call Pattern of Reflection Enc. 28
Table 4.23: Family API Call Pattern of Reflection Enc. 28
Table 4.24: An API Call Pattern of Class Enc. 29
Table 4.25: Family API Call Pattern of Class Enc. 29
Table 4.26: An API Call Pattern of Trivial+String Enc. 30
Table 4.27: Family API Call Pattern of Trivial+String Enc. 30
Table 4.28: An API Call Pattern of Trivial+String+Reflection Enc. 31
Table 4.29: Family API Call Pattern of Trivial+String+Reflection Enc. 31
Table 4.30: An API Call Pattern of Trivial+String+Reflection+Class Enc. 32
Table 4.31: Family API Call Pattern of Trivial+String+Reflection+Class Enc. 32
Table 4.32: Shortened Name for System Command 33
Table 4.33: A System Command Pattern of Trivial Enc. 34
Table 4.34: Family System Command Pattern of Trivial Enc. 34

Table 4.35: A System Command Pattern of String Enc. 35
Table 4.36: Family System Command Pattern of String Enc. 35
Table 4.37: A System Command Pattern of Reflection Enc. 36
Table 4.38: Family System Command Pattern of Reflection Enc. 36
Table 4.39: A System Command Pattern of Class Enc. 37
Table 4.40: Family System Command Pattern of Class Enc. 37
Table 4.41: A System Command Pattern of Trivial+String Enc. 38
Table 4.42: Family System Command Pattern of Trivial+String Enc. 38
Table 4.43: A System Command Pattern of Trivial+String+Reflection Enc. 39
Table 4.44: Family System Command Pattern of Trivial+String+Reflection Enc. 39
Table 4.45: A System Command Pattern of Trivial+String+Reflection+Class Enc. 40
Table 4.46: Family System Command Pattern of Trivial+String+Reflection+Class Enc. 40
Table 4.47: Shortened Name for Intent 40
Table 4.48: An Intent Pattern of Trivial Enc. 41
Table 4.49: Family Intent Pattern of Trivial Enc. 41
Table 4.50: An Intent Pattern of String Enc. 42
Table 4.51: Family Intent Pattern of String Enc. 42
Table 4.52: An Intent Pattern of Reflection Enc. 43
Table 4.53: Family Intent Pattern of Reflection Enc. 43
Table 4.54: An Intent Pattern of Class Enc. 44
Table 4.55: Family Intent Pattern of Class Enc. 44
Table 4.56: An Intent Pattern of Trivial+String Enc. 45
Table 4.57: Family Intent Pattern of Trivial+String Enc. 45
Table 4.58: An Intent Pattern of Trivial+String+Reflection Enc. 46
Table 4.59: Family Intent Pattern of Trivial+String+Reflection Enc. 46
Table 4.60: An Intent Pattern of Trivial+String+Reflection+Class Enc. 47
Table 4.61: Family Intent Pattern of Trivial+String+Reflection+Class Enc. 47
Table 4.62: Shortened Name for App Component 48
Table 4.63: An App Component Pattern of Trivial Enc. 49
Table 4.64: Family App Component Pattern of Trivial Enc. 49
Table 4.65: An App Component Pattern of String Enc. 50
Table 4.66: Family App Component Pattern of String Enc. 50
Table 4.67: An App Component Pattern of Reflection Enc. 51
Table 4.68: Family App Component Pattern of Reflection Enc. 51
Table 4.69: An App Component Pattern of Class Enc. 52
Table 4.70: Family App Component Pattern of Class Enc. 52
Table 4.71: An App Component Pattern of Trivial+String Enc. 53
Table 4.72: Family App Component Pattern of Trivial+String Enc. 53
Table 4.73: An App Component Pattern of Trivial+String+Reflection Enc. 54
Table 4.74: Family App Component Pattern of Trivial+String+Reflection Enc. 54

Table 4.75: An App Component Pattern of Trivial+String+Reflection+Class Enc. 55
Table 4.76: A Family App Component Pattern of Trivial+String+Reflection+Class Enc. 55

LIST OF FIGURES

Figure 3.1: Features 14
Figure 3.2: Research Methodology 16
Figure 4.1: Permission Found from Trivial Enc. 18
Figure 4.2: Permission Found from String Enc. 19
Figure 4.3: Permission Found from Reflection Enc. 20
Figure 4.4: Permission Found from Class Enc. 21
Figure 4.5: Permission Found from Trivial+String Enc. 22
Figure 4.6: Permission Found from Trivial+String+Reflection Enc. 23
Figure 4.7: Permission Found from Trivial+String+Reflection+Class Enc. 24
Figure 4.8: Suspicious API Call Found from Trivial Enc. 26
Figure 4.9: Suspicious API Call Found from String Enc. 27
Figure 4.10: Suspicious API Call Found from Reflection Enc. 28
Figure 4.11: Suspicious API Call Found from Class Enc. 29
Figure 4.12: Suspicious API Call Found from Trivial+String Enc. 30
Figure 4.13: Suspicious API Call Found from Trivial+String+Reflection Enc. 31
Figure 4.14: Suspicious API Call Found from Trivial+String+Reflection+Class Enc. 32
Figure 4.15: System Command Found from Trivial Enc. 33
Figure 4.16: System Command Found from String Enc. 34
Figure 4.17: System Command Found from Reflection Enc. 35
Figure 4.18: System Command Found from Class Enc. 36
Figure 4.19: System Command Found from Trivial+String Enc. 37
Figure 4.20: System Command Found from Trivial+String+Reflection Enc. 38
Figure 4.21: System Command Found from Trivial+String+Reflection+Class Enc. 39
Figure 4.22: Intent Filter Found from Trivial Enc. 41
Figure 4.23: Intent Filter Found from String Enc. 42
Figure 4.24: Intent Filter Found from Reflection Enc. 43
Figure 4.25: Intent Filter Found from Class Enc. 44
Figure 4.26: Intent Filter Found from Trivial+String Enc. 45
Figure 4.27: Intent Filter Found from Trivial+String+Reflection Enc. 46
Figure 4.28: Intent Filter Found from Trivial+String+Reflection+Class Enc. 47
Figure 4.29: App Component Found from Trivial Enc. 48
Figure 4.30: App Component Found from String Enc. 49
Figure 4.31: App Component Found from Reflection Enc. 50
Figure 4.32: App Component Found from Class Enc. 51
Figure 4.33: App Component Found from Trivial+String Enc. 52
Figure 4.34: App Component Found from Trivial+String+Reflection Enc. 53
Figure 4.35: App Component Found from Trivial+String+Reflection+Class Enc. 54

ABSTRACT

Android smartphone's security and privacy of personal information remain threatened because of popularity. Noxious applications represent a danger to the security of the Android. Yet understanding Android malware utilizing dynamic examination can give a far-reaching view, it is still exposed to surprising expense in condition arrangement and manual endeavors in examination. To classify or detect android malware applications, it is important to identify pattern of malware. In this study, some important static features pattern of obfuscated android malware applications has been proposed. AndroShow, a broad static analysis-based feature analyzer is introduced that identifies important features pattern of Android. Permission, API call, app component, intent filter and system call patterns are embedded in vector matrix. In order to classification and detection of android malware application this malware pattern analysis will beneficial. AndroShow investigate 10479 obfuscated malware applications. These malware applications consist of seven categories of obfuscation techniques taken from PRAGuard dataset.

CHAPTER 1
INTRODUCTION

1.1 Background Study

Android is a smartphone operating system developed by Google, introduced in 2008. Now it is most popular operating system than other platforms. Android smartphones have persistently been supplanting the customary cell phones. Smartphone have been changing activity of day-to-day to people. Now, many desktop tasks can be completed by smartphone. Hotel room booking, airplane ticket, bus ticket, online banking, online marketing and so many important tasks are being completed “on the go”. These demandable tasks playing an important role to increase smartphone users. Monthly, there are over 2 billion active android devices around the world (Ben, 2017). According to International Data Corporation reports on smartphone market share, android system has 86.8% of the smartphone operating systems, up to the third quarter of 2018 (IDC). Besides, there are over 2 million apps in the app store (Statista). Smartphone user keeps personal, business information in the device which are very much sensitive. Therefore, smartphones running android are progressively focused by aggressors and tainted with vindictive programming. As opposed to different portion, android takes into account introducing applications from unsubstantiated sources. For example, outsider markets, which makes packaging and disseminating applications with malware simple for aggressors (Arp et al., 2014). (Fereidooni et al., 2016) mentioned that in the principal half of 2014, F-Secure detailed that new risk families or 295 new variations of realized families were gathered. it merits referencing that 294 out of these 295 families keep running on Android. In addition, they also referred that in the main quarter of 2015, Kaspersky mobile security distinguished 103, 072 new perilous applications, a three-overlap increment from last quarter of 2014. Obviously quick and dependable components are required to distinguish and investigate possibly perilous applications (Suarez-Tangil et al., 2017)

1.2 Motivation of the Research

This part studies the three factors that motivate the research become involved in this bachelor study. First, Pathao is a popular transportation company in Bangladesh. It’s local ride sharing app is Pathao. This app purportedly protects clients' personal messages abusing the information security control. The versatile application-based vehicle hailing administration keeps all information including individual discussion at inbox including contacts which is out of protection controls, affirmed a client demonstrating video archives. An ongoing video via web-based networking media demonstrates that the application protects all contacts and individual messages to structure client administrations. Pathao specialists, be that as it may, precluded the claim from securing breaking client information (Jannatul, 2018). It requires dangerous permissions like read sms, read contacts. These permissions are related to user’s privacy. Second, lack of study in malware pattern analysis. Third, self-study in obfuscated android malware application.

1.3 Problem Statement

There are several many android malware trends exist in global aspect. For these trends, smartphone device security is compromising. Three latest trends are undertaken for problem statement. First, aggressive ads, it's can (1) haphazardly promotions springing up, (2) leaks of private data. It requires dangerous permissions like permission of read and write to default browser, external code executes like DexClassLoader. Second, lockers, so called ransomware. It's (1) encrypt device data (2) show ransom note in lock screen (3) request payment in cryptocurrencies. It requires System_Alert permission, also it can add new device administrator as well as delete files and so many. Third, bankers, the quickest developing category. Some characteristics are (1) persuading phishing assaults to bait clients into giving bank data, (2) very much perilous on the grounds that may have coordinate cost affect, (3) typically extremely advanced and complex malware (Jan & Ondrej, 2018).

1.4 Research Question

How does malware pattern analysis of different strategies beneficial for android malware detection?

1.5 Research Objective

To identify malware pattern for detecting android malware application.

1.6 Research Contribution

In this paper, several works have been done. Main contribution of this paper analysis is given below -

- Static analysis has been performed on obfuscated Android malware application.
- Analysis performs on five features - permission, API call, intent filter, app component, system call.
- Features pattern proposed in 2D matrix. Where column name is the feature tag name and rows are the 0/1 with family name.
- Most uses features demonstrate in 2D bar chart.
- Features extracted from obfuscated malware dataset, PRAGuard. This dataset contains 10,479 obfuscated malware applications with seven different obfuscation techniques.

- Every obfuscated technique wise feature pattern has been proposed.

1.7 Thesis Organization

Following chapters are sort out as: literature review is discussed in chapter 2. Research methodology that contains research method, tools, environment that are used in this paper work, discussed in chapter 3. Results and discussions of analysis in chapter 4. Chapter 5 conclude the paper with recommendation, finally.

CHAPTER 2
LITERATURE REVIEW

2.1 Previous Study

(Arp et al., 2014) introduced DREBIN, a lightweight strategy for discovery of android malware that empowers recognizing pernicious applications specifically on the smartphone. It performs a wide static investigation, assembling number of highlighted features of an application as could reasonably be expected. They extract eight set of features but more specifically to robust analysis they extracted permissions, API calls, intent filters, network addresses. They implant these features in joint vector space so that classical patterns characteristics for malware can be simply identified. They evaluate 123,453 applications and 5560 malware samples in DREBIN and get a detection rate of 94% malware with false positive rate of 1%. It's similar to 1 false alarm in 100 applications.

(Suarez-Tangil et al., 2017) presented DroidSieve, an android malware classifier dependent on static examination that is quick, exact, and flexible to obfuscation. It's depends on diverse features that are known to be characteristic of android malware, covering code structure, permissions, set of invoked components and API calls. DroidSieve plays out a novel profound assessment of the application to recognize separating highlights missed by existing strategies, counting local components, obfuscation relics, and features that are invariant under obfuscation. For malware recognition, they accomplish up to 99.82% precision with zero false positives and for family detection of obfuscated malware, they accomplish 99.26% accuracy.

(Iqbal & Zulkernine, 2018) implement SpyDroid, a multiple real time detection tool for android malware. This detection framework deploys multiple sub-detectors on real device. These sub-detectors monitor the app behavior and then report to SpyDroid detector. After analyzing the report, it takes decision the app malicious or not. On classifying 4956 apps including 2711 malware and 2254 malware samples different sub-detectors classify same application differently. But ensemble of sub-detectors raises the identification rate remarkably.

A graph-based model applied by (Alasmary et al., 2019) to detect android malware. They name it Poster. They analyze Android and IoT malware to understand the characteristics. They conduct a depth analysis on graph properties binaries of Control Flow Graph (CFG) structure. General characteristics and graph algorithmic properties were identified from 2874 benign and 2891 malware apps. They achieved best and highest accuracy rate 97.9%, FPR (1.1%), FNR (11.6%) with Random Forest. They apply different machine learning classifier like LR, SVM, RF and CNN deep learning methods on the entire dataset.

A dynamic zero-day Android malware detection system presented by (Grace et al., 2012). Authors developed an automated system called RiskRanker. It examines whether a specific app disclose dangerous behavior like initiating a root exploit or transferring background SMS. They perform analysis on 118,318 apps to get effectiveness and accuracy. Their system uncovers 718 malware samples in 29 families and 320 zero-day malware apps from 11 well defined families.

Remarkable anomalies of network behavior of smartphone applications are planned in (Murtaz et al., 2018). The authors developed a system whose main goal is to protect smartphone user's device and cellular infrastructure corporations from malicious apps from nine traffic feature measurements. They get a common accuracy of 94% for five machine learning classifiers; Random Forest, K-Nearest Neighbor, Decision Tree, Random Tree, and Regression. The model uses cluster strategies containing stream based, bundle based and time-based highlights to narrate malware families.

(Parker et al., 2018) inspect whether a data mining technique initially developed to detect malware on a Windows operating system can be deployed to detect malware in Android devices. They proposed a novel algorithm that depends on step sizes and a popularized multi-layer vector space (MLVC) model for detecting Android malware. They compare the effectiveness of two techniques and get a result that shows two methods are able to accurately classify the samples as malware or benign with powerful precision.

(Chang et al., 2018) demonstrate malware family's main characteristic operations or activities mainly related to its intent. They introduced ANTSdroid, a novel automatic dynamic Android profiling system. They applied Runtime API sequence Motif Mining Algorithm (RasMMA) based on the analysis of the sensitive and permission -related execution traces of the threads and processes of a set of variant APKs of a malware family. They take 10 families of 2568 malware samples from the DREBIN dataset. DroidKungFu malware family samples used to demonstrate the generated family signature actually captures key sample activities of the family. The investigation results reveal the usefulness of using the generated family signature to detect new variants using real-world datasets.

(Martín et al., 2019) proposed a machine learning based android malware detection based on signature. They inspect a large collection of Android applications (>80K) marked as malware by at least on AV from a set of 61 unlike engines, allowing almost 260K malware signatures. They perform a depth analysis on malware families and their interrelations engine wide, distinguishing up to 41 unlike malware families which belong to three large categories; Adware, Harmful and Unknown. They classify Unknown applications by machine learning classification tools; Logistic Regression with Lasso regularization and Random Forest for placing into Adware or Harmful. These classifier models yield outstanding classification result (F1-score of 0.84) and shows some of Unknown family classes into either Adware/Harmful threat.

Machine learning based static analysis proposed by (Tiwari & Shukla, 2018). Authors analyze API and permission for malware detection in android device. Their dataset consists of 669 malware samples and 652 benign samples. They got 97.25% accuracy with the use of logistic regression and 96.21% of accuracy with the support vector machine further without any pre-sequence of dataset. After preprocessing the dataset, they obtained 97.72% accuracy with 350 features and 94.69% accuracy with 30 features using support vector machine. They use only common features instead of a large number of features and further optimized the feature using PCA and obtained 94.31% of accuracy with 30 features.

A deep Autoencoder model is proposed by (He et al., 2018). It is a neural network model implemented with Google’s open source TensorFlow deep learning library. It is designed to reduce the dimension of feature vectors. They also applied logistic regression model to learn and classify the Android applications to be normal or not. They experiment on 5000 normal applications and 1200 abnormal applications and get recall rate and F1 value respectively 0.93 and 0.643.

(Qamar et al., 2019) reveals all endeavors towards versatile malware creation, spreads, dispersal and identification. The all-around characterized scientific classifications are extensively introduced and examined the need to crumble its unsafe effect on network. Besides, in this paper measurable examination and the exploration work led amid years 2013-2019 in the space of versatile malware investigation alongside man-made consciousness recognition systems are talked about. They propose some future headings for analysts that grows increasingly precise, productive, strong and versatile system in context of android malware identification. they additionally moderately characterize and look at avoidance systems, for example, polymorphism, java reflection, muddling and control stream change that is utilized by malware creators to sidestep from detection. Additionally, authors expressly decide a few shortcomings from existing methodologies that are not altogether examined by researchers. To adapt to this developing risk, difficulties and future work headings are likewise showed to give a brisk rule to the scholarly community and industry alike.

Before study of (Hu et al., 2019), no existing studies have investigated the characteristics of money-making apps. They uncover a few fascinating perceptions: (1) cash making applications have turned into the objective of malware designers, as we discovered a considerable lot of them open versatile clients to genuine protection and security dangers. Generally, 26% of the examined applications are conceivably malignant. (2) these applications have pulled in a huge number of clients, be that as it may, numerous clients grumble that they are swindled by these applications. We likewise uncovered that positioning misrepresentation procedures are generally utilized in these applications to advance the positioning of applications inside application markets. (3) these applications generally spread unseemly and malignant substance, while unsuspicious clients could get contaminated. Authors exact examination results uncover different terrible substance, for example, malware, conflicting substance and substance with forceful promotions. Study reveals that "Content Sharing" applications and "Pay-Per-Install" applications have gotten the greatest number of client grumblings and most conceivably malevolent applications have a place with these two classes. Most number of client grumblings and most possibly pernicious applications have a place with these two classes. Authors investigation has uncovered different fascinating discoveries, including the nearness of positioning misrepresentation, protection issues, malware nearness, conflicting and pernicious appropriated substance.

2.2 FEATURE ENGINEERING

2.2.1 Permission

Permission plays an important role in Android apps. A huge piece of Android's worked in security is its permissions framework (Fereidooni et al., 2016). Protecting privacy of Android user is most important in online world. This important task is done by permission. To access sensitive user data (Contacts, SMS), moreover explicit system features (Camera, Location) permission must be requested by android apps. Based on feature, system allow the permission automatically or might provoke the user to allow the appeal. All permission present publicly in <uses-permission> tags in the manifest file. Android app that requires normal permission (do not harm to user's privacy or device operation) system automatically allow these permissions to app. App that requires dangerous permission (permission that can harmful for user's privacy or device normal operation) the user must explicitly allow to accept those permissions (Android Developers, a). Permissions enable an application to get to possibly perilous API calls. Numerous applications need a few authorizations to work appropriately and client must acknowledge them at install time. Permission gives a more top to bottom view on the functional qualities of an application. Malware authors include dangerous permission in manifest that is not relevant to app and also declare much more permissions than literally required (Felt et al., 2011, Johnson et al., 2012). Therefore, it is become more difficult to detect malicious application based on permissions.

Ref.	Features	Samples	Accuracy	Machine Learning Methods
(Arslan et al., 2019)	Permission	7400	91.95%	BayesNet, Naive Bayes, LogisticRegression, K-star, OneR, Multilayer Perceptron, K-nearest, Decision Tree, J48, Random Forest
(Dighe et al.)	Permission	3784	94.50%	J48, Random Forest
(Huang et al., 2013)	Permission	125,249	81%	AdaBoost, Naïve Bayes, C4.5, SVM
(Kumar et al., 2019)	Permission	11,752	98.1%	Random Forest
(Aung & Zaw, 2013)	Permission	700	91.75%	K-Means, J48, RF, CART
(Dong, 2017)	Permission	28,847	95.1%	Linear model, Tree Model, Neural Network and Ensemble Model

Table 2.1: Some Previous Works on Permission

2.2.2 Intent Filter

An Intent is an informing object you can use to ask for an operation from another application component. Despite the fact that intent makes easier communication between components in a few different ways, there are three basic ways (1) starting an activity (2) starting a service (3) delivering a broadcast. Two types of intent are there (1) Explicit Intents (2) Implicit Intents (Android Developers, b). Explicit Intents identify the components to start with by containing targeted package names and class names. Normally, Explicit Intents are utilized to interface parts inside a similar application and intended for inter application communications. In contrast to Explicit Intents, Implicit Intents do not name a particular segment, however rather proclaim general activities to perform. At the point when an application makes an Implicit Intent, the Android framework finds the suitable segment to begin by contrasting the substance (i.e., action, category and data) of the Intent to the pronounced Intent Filters. On the off chance that the Intent matches an Intent Filter, the framework begins that segment and conveys it the Implicit Intent item (Xu et al., 2016). If multiple Intent filters are matches than system shows a dialog box to user to pick up which app to use. An Intent filter is a declaration in an app's manifest.xml files that states the type of intents of the component will receive. Suppose, an activity declares an intent filter, means that other apps can directly start the activity with an undoubtable type of intent. Similarly, if an activity does not declare an intent, then it can be activated only by Explicit Intent (Android Developers, b). Intent used in inter component and inter app communication. Intent filters identify a particular access for a component as well as the application. Intent filters can be used for spying specific intents. Malware is responsive to particular set of system events. So, Intent filters can be indicator.

Ref.	Features	Samples	Accuracy/Findings	Methods
(Elish et al., 2015)	Intent	2644	Shows effective solution need to detect collusion attack	Static Analysis
(Xu et al., 2015)	Intent	17,290	97.4%	SVM
(Feng et al., 2014)	Intent	-	Resilience to some obfuscation techniques in detection	Call Graph, Taint Analysis
(Feizollah et al., 2017)	Permission, Network, Intent	7406	95.5%	Bayesian K2, Geneticsearch, HillClimber, LAGDHillClimber

(Li et al., 2015)	Intent	2283	96.6%	Taint Analysis, Data Flow Analysis
(Li et al., 2014)	Intent	2000	75%	Taint Analysis, CFG, Data Flow Analysis

Table 2.2: Existing Researches on Intent Filter

2.2.3 API Call

API stands for Application Programming Interface. In simple terms, APIs simply enable applications to speak with each other. Envision the accompanying situation: You (as in, your application, or your customer, this could be an internet browser) needs to get to another application's information or usefulness. For instance, maybe you need to get to all Twitter tweets that notice the #malware hashtag. You could email Twitter and request a spreadsheet of every one of these tweets. In any case, at that point you'd need to figure out how to bring that spreadsheet into your application; and, regardless of whether you put away them in a database, as we have been, the information would end up obsolete in all respects rapidly. It is difficult to stay up with the latest. It would be better and easier for Twitter to give you an approach to question their application to get that information, so you can view or utilize it in your own application. It would remain state-of-the-art consequently that way (Perry, 2017). API includes of principle set of packages and classes. Most apps use large number of API calls, so it's help us to characterize and differentiate malware from benign apps. (Peiravian & Zhu, 2013) state that benign apps use most APIs than malware apps. The author's in (Seo et al., 2014) has listed some suspicious API calls used by malware applications. For example - sendTextMessage, getPackageManager, getDeviceId, Runtime.exec.

Ref.	Features	Samples	Accuracy / Findings	Methods
(Peiravian & Zhu, 2013)	Permission, API Call	2510	96.39%	SVM, J48, Bagging
(Yang et al., 2017)	Permission, API Call	28558	84.9% [Offline] 99.0% [Online]	String Subsequence Based SVM
(Skovoroda & Gamayunov, 2017)	Permission, API Call	10449	90-94%	Static, Model Matching

(Aafer et al., 2013)	API Call	20000	99.0%	ID3, C4.5, KNN, SVM
(Shen et al., 2018)	API Call	8598	97.6% TP 91.0% TN	Complex Flow, N-gram Analysis
(Ghani et al., 2015)	API Call, Manager Class	-	SMSManager, Telephony Manager most used in malware	Static Analysis, Feature Comparison

Table 2.3: Existing Researches on API Call

2.2.4 System Call

Android core is the modified version of Linux 2.6 kernel. For adopting mobile operating system devices this modification was done. The Android Kernel explicit bit upgrades on power management, shared memory drivers, alert drivers, folios, bit debugger and lumberjack and low memory executioners. System calls connect Android application and kernel. Whenever a client asks for administrations like call a telephone in client mode through the telephone call application, the demand is sent to the Telephone Directory Service in the application structure. The Dalvik Virtual Machine in Android runtime changes the client ask for gone by the Telephone Manager Service to library calls, which results in various framework calls to Android Kernel. While executing the system call, there is a change from client mode to part mode to play out the delicate activities. At the point when the execution of activities asked for by the system call is finished, the control is come back to the client mode (Malik & Khatter, 2016). As talked about over, the system calls are the communicator between the client and the bit. This implies all solicitations from the applications will go through the System Call Interface before its execution through the equipment. So, catching and dissecting the system call can give data about the conduct of the application. (Seo et al., 2014) listed some system calls that are often used in malware applications. for example - chmod, su, mount, sh, killall, reboot, mkdir, ln, ps.

Ref.	Features	Samples	Accuracy / Findings	Methods
(Dimjaševic et al., 2015)	System Call	12,660	93%	SVM, RF, LASSO, Ridge Regularization
(Firdaus & Anuar, 2015)	System Call, Directory Path, Code Based	1100	92.5%	Multilayer Perceptron, Random Forest, Naïve Bayes

(Da et al., 2016)	System Call	152	>93.0%	Random Forest
(Kedziora et al., 2018)	Broadcast-Receiver, System Call, API Call	1958	80.3% - 80.7%	RF, SVM, K-NN, Naive Bayes, Logistic Regression
(Tchakounté & Dayang, 2013)	System Call	Malgenom (DroidDream)	Click Event Perform Malicious Tasks	Dynamic Analysis
(Wahanggara & Prayudi, 2015)	System Call	460	90.0% (Polynomial Kernel) 86.0% (RBF Kernel)	SVM
(Malik & Khatter, 2016)	System Call	645	Malware App Invokes System Calls More Frequently Than Benign App	Dynamic Analysis

Table 2.4: Existing Researches on System Call

2.2.5 App Component

Application components are the fundamental structure squares of an Android application. Every component is a section point through which the framework or a client can enter your application (Android Developers, c). These parts are inexactly coupled by the application show record AndroidManifest.xml that depicts every segment of the application and how they connect (tutorialspoint). Some of them depend on others. There are following four types of component used in Android application -

- Activities
  An action is the section point for communicating with the client. It speaks to a solitary screen with a UI. For instance, an email application may have one movement that demonstrates a rundown of new messages, another action to create an email, and another action for perusing messages. In spite of the fact that the exercises cooperate to shape a durable client involvement in the email application, everyone is free of the others. All things considered; an alternate application can begin any of these exercises if the email application permits it. For instance, a camera application can begin the movement in the email application that makes new mail to enable the client to share an image.

- Services
  An administration is a broadly useful section point for keeping an application running out of sight for a wide range of reasons. An administration is a part that keeps running out of sight to perform long-running activities. An administration does not give a UI. For instance, an administration may play music out of sight while the client is in an alternate application, or it may get information over the system without blocking client connection with a movement.
- Broadcast receivers
  A broadcast receiver is a part that empowers the framework to convey occasions to the application outside of standard client stream, permitting the application to react to framework wide communicate declarations. Since communicate collectors are another all-around characterized section into the application, the framework can convey communicates even to applications that aren't as of now running. Thus, for instance, an application can plan an alert to present a warning on enlightening the client regarding an up and coming occasion... also, by conveying that caution to a BroadcastReceiver of the application, there is no requirement for the application to stay running until the alert goes off. In spite of the fact that communicate recipients don't show a UI, they may make a status bar warning to alarm the client when a communicate occasion happens (Android Developers, c).
- Content providers
  A content provider part supplies information from one application to others on solicitation. The information might be put away in the record framework, the database or elsewhere altogether. Through the content provider, different applications can inquiry or change the information if the content provider permits it. For instance, the Android framework gives a content provider that deals with the client's contact data. Content provider are additionally helpful for perusing and composing information that is private to your application and not shared.

Ref.	Features	Samples	Accuracy / Findings	Methods
(Wu et al., 2012)	App Component etc.	1738	97.87%	Singular Value Decomposition KNN
(Wang et al., 2017)	App Component etc.	8385	99.7%	KNN, Random Forest, J48
(Kim et al., 2019)	App component etc.	35,331	98.0%	MNN-z, MNN-s, DNN

(Shen et al., 2014)	App Component etc.	308	86.36%	Topology Graph
(Li et al., 2018)	App Component etc.	19,000	99.01% (From DREBIN) 99.2% (From AMD)	Factorization Machine
(Rana et al., 2018)	App Component etc.	11120	94.0%	RF, DT, EDT, GB, SVM, NN-MLP, NB, k-NN, DA, LR, BAGG, KMN

Table 2.5: Existing Researches on App Component

CHAPTER 3
RESEARCH METHODOLOGY

3.1 Dataset

This paper is based on Android PRAGuard dataset that was proposed by (Maiorca et al., 2015). It has 10479 malware samples. This dataset was established by obfuscating malware samples of the MalGenome and Contagio Minidump datasets with seven different obfuscation techniques such as string encryption, class encryption, reflection etc.

3.2 Feature

This paper's aim is extracting most important features of Android that are often used by malware writers. To grant for a gentile and tensile analysis, this paper illustrates all extracted features such as permissions, app components, API calls, intent filters, system calls based on (Arp et al., 2014).

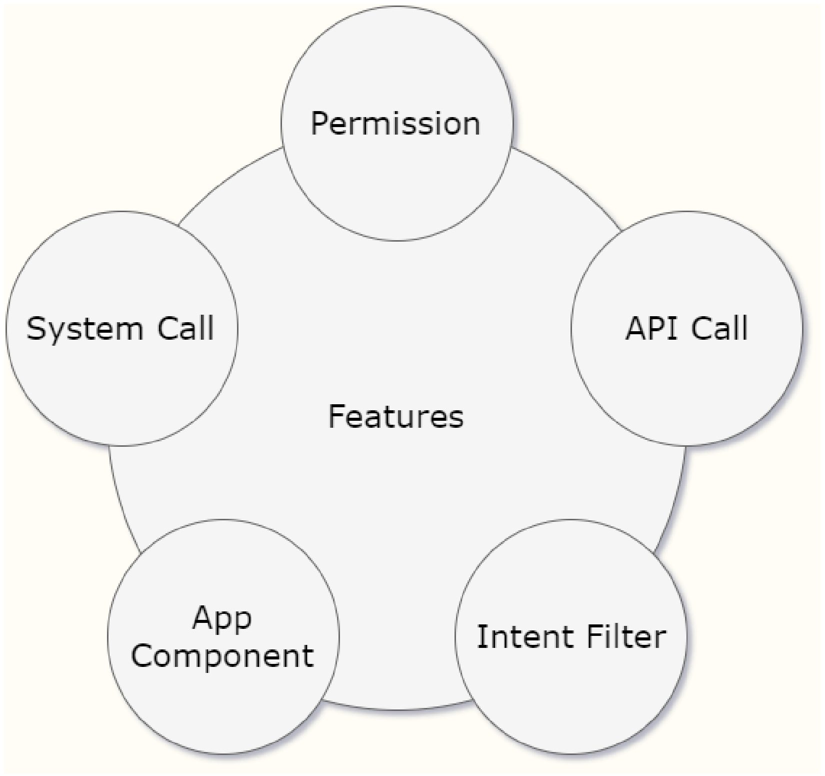


Figure 3.1: Features

3.3 Tool

This paper needs to analyze the details of APK. To explore such thing is called reverse engineering. Reverse engineering can be done with Androguard. Androguard is a tool for playing with android files which is developed in python done by VirusTotal project. What can play Androguard that’s below:

- DEX / ODEX
- APK
- Android’s binary xml
- Android resources
- Disassemble DEX/ODEX bytecodes
- Decompiler for DEX/ODEX files

CLI or graphical UI can be used for Androguard. Besides, as a library Androguard is adaptable. This analysis uses Androguard as library or module (in python) to analyze DEX and APK files.

3.4 Environment

Collected dataset has seven obfuscation techniques along with 10479 malware samples. Androguard is a pretty much big library. Memory and CPU usage will be high while extracting information from dataset by Androguard. For comfort and efficient analysis this paper adopts HP i5 2.30 GHz 8GB computing environment. OS: Windows 10, language: Python 3, Module: CSV, Matplotlib.

3.5 Procedure

Every work has predefined process. So therefore, this paper works has some to gain paper's objective. Working procedure of this paper analysis demonstrated below -

A. At first step, AndroShow inspect all 10479 malware apk samples whether they are valid or not. Because of using obfuscation technique some apk’s structure might be broken Maiorca et al. [7]. Therefore, it’s needs to examine first. So, Androguard could not extract some apk’s. These apk’s are not in proper zip format.
B. Second step is extracting features (3.2) using Androguard. Androguard has command line interface. At a time only one apk can extracted. So, extracting many apks are not efficient with CLI. For this situation AndroShow has been introduced. It uses Androguard library. AndroShow can extract multiple apks at a time.

C. In this step CSV file is created upon on features tag name got from second step. These features are columns name and rows are 0/1. If a feature found in apk than its 1 otherwise 0.
D. After that, creating a 2D bar chart with matplotlib from CSV file where X-axis is for features (3.2) tag name and Y-axis for total counted number of features (3.2) tag found from CSV, counting on how many 1 found from individual column related to feature tag name.
E. Five and last is to get most uses features of individual obfuscated techniques from bar chart.

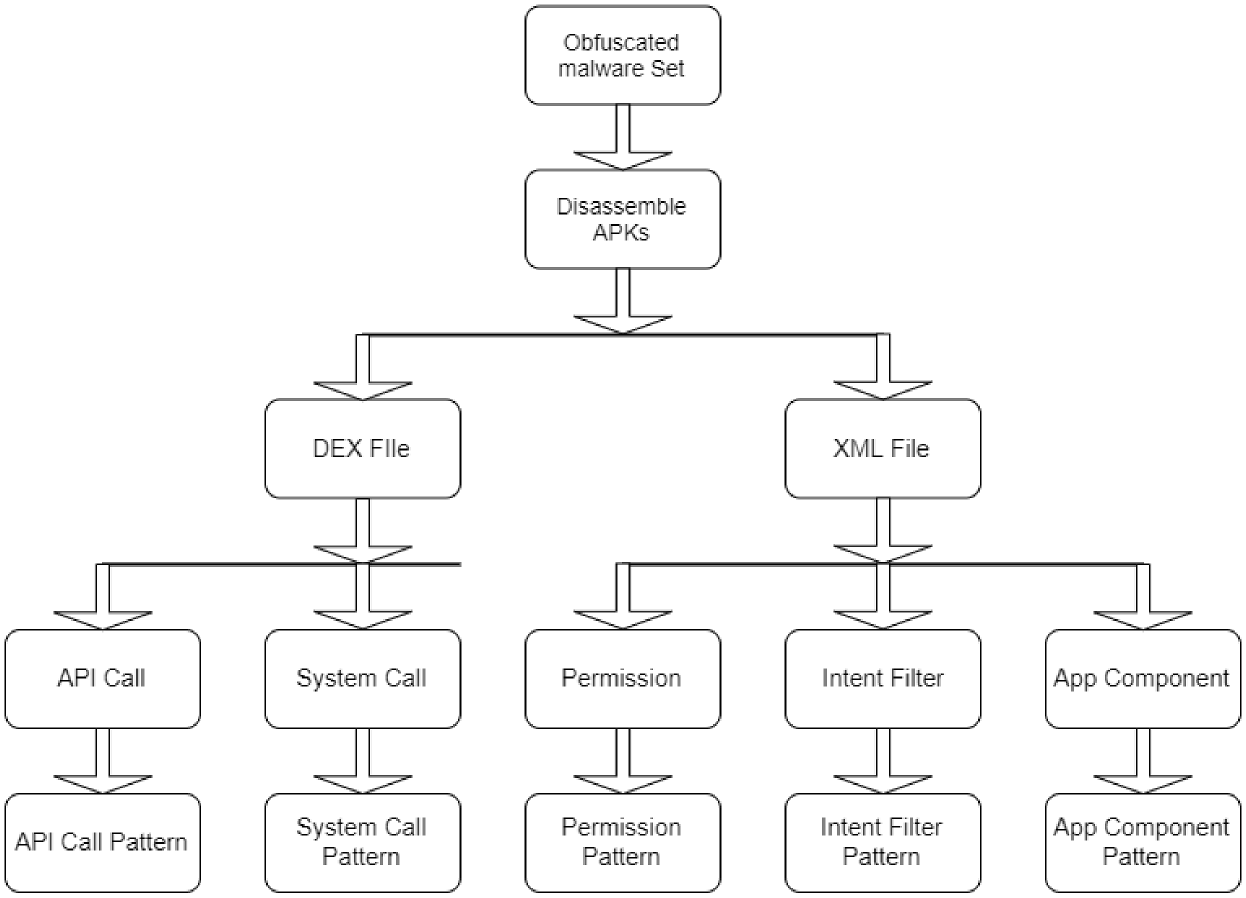


Figure 3.2: Research Methodology

CHAPTER 4
RESULT AND DISCUSSION

AndroShow inspect a broad static analysis of APK. It extracts all features (3.2) from PRAGuard dataset. In this following section this paper will shows every features single and possible combinations pattern of seven obfuscated techniques as well as gives a detailed overview of analysis.

Full	Short
Permission	PR
Requested Permission	RPR
API Call	APC
App Component	AC
Intent	IN
System Command	COM

Table 4.1: Short Version of Features

4.1 PERMISSION ANALYSIS

In this section, this paper illustrates every obfuscated technique permission and requested permission analysis report. Besides, every techniques permission pattern and technique wise a malware family pattern is given.

Full Version	Short Version
INTERNET	PR1
READ_PHONE_STATE	PR2
READ_CONTACTS	PR3
ACCESS_NETWORK_STATE	PR4
SEND_SMS	PR5
WRITE_EXTERNAL_STORAGE	PR6

Table 4.2: Shortened Name for Permissions

4.1.1 Trivial Encryption

From Trivial enc. analysis 152 permissions including 18 requested permissions found. Top 50 permissions are displaying below -

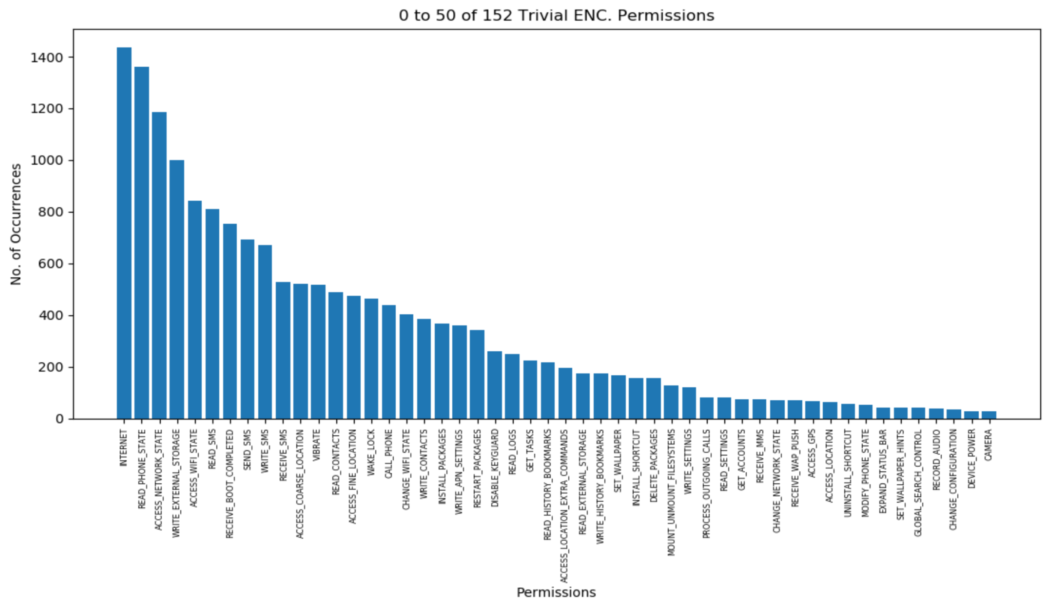


Figure 4.1: Permission Found from Trivial Enc.

PR1	PR2	PR3	-	PR4	PR5	PR6
1	1	0	-	1	1	1

Table 4.3: A Permission Pattern of Trivial Enc.

Family	PR1	PR2	PR3	-	PR4	PR5	PR6
jSMSHider	1	1	1	-	1	0	0
jSMSHider	1	1	1	-	1	0	0
AnserverBot	1	1	1	-	1	1	0
AnserverBot	1	1	1	-	1	1	0
Geinimi	1	1	1	-	0	1	1
Geinimi	1	1	1	-	1	1	1

Table 4.4: Family Permission Pattern of Trivial Enc.

4.1.2 String Encryption

Like Trivial enc., same number of normal permissions and requested permissions found from String enc. Top 50 are showing below -

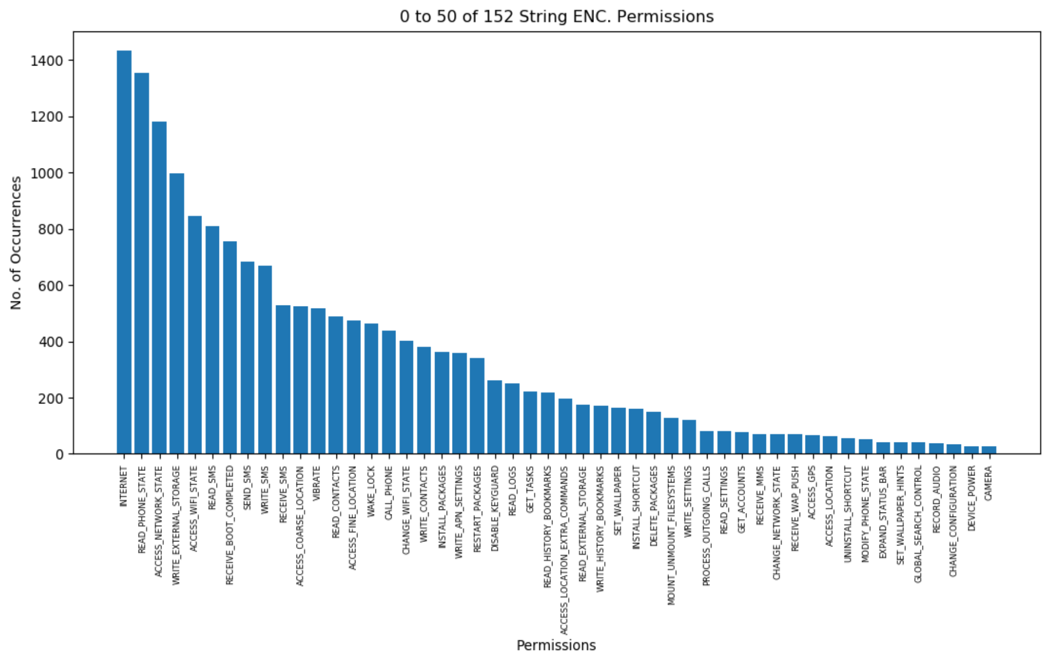


Figure 4.2: Permission Found from String Enc.

PR1	PR2	PR3	-	PR4	PR5	PR6
1	1	1	-	0	0	0

Table 4.5: A Permission Pattern of String Enc.

Family	PR1	PR2	PR3	-	PR4	PR5	PR6
DroidKungFu1	1	1	0	-	1	0	1
DroidKungFu1	1	1	0	-	1	0	1
KMin	1	1	1	-	1	1	1
KMin	1	1	1	-	1	1	1
Plankton	1	1	1	-	0	0	1
Plankton	1	1	1	-	0	0	0

Table 4.6: Family Permission Pattern of String Enc.

4.1.3 Reflection Encryption

Alike with above two techniques, Reflection enc. have also same findings. Figure 4.3 shows top 50 features of Reflection enc.

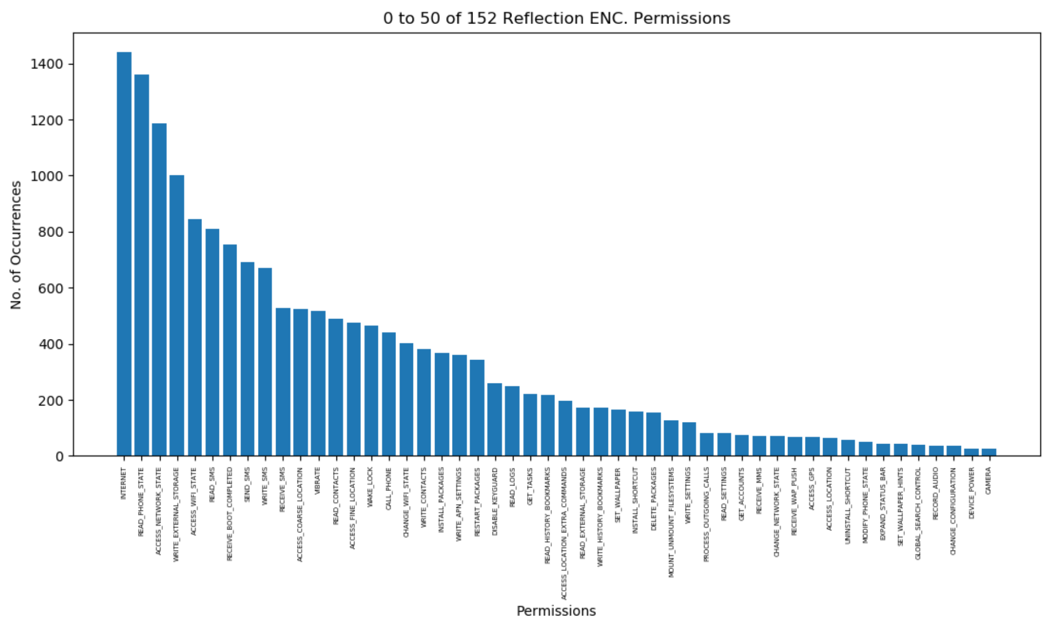


Figure 4.3: Permission Found from Reflection Enc.

PR1	PR2	PR3	-	PR4	PR5	PR6
1	1	0	-	1	0	1

Table 4.7: A Permission Pattern of Reflection Enc.

Family	PR1	PR2	PR3	-	PR4	PR5	PR6
BaseBridge	1	1	0	-	1	0	1
BaseBridge	1	1	0	-	1	0	1
DroidKungFu3	1	1	0	-	1	0	1
DroidKungFu3	1	1	1	-	1	0	1
ADRD	1	1	0	-	1	0	1
ADRD	1	1	0	-	1	0	1

Table 4.8: Family Permission Pattern of Reflection Enc.

4.1.4 Class Encryption

Similarly, upon three techniques number of permission and requested permissions are equivalent to Class enc. Top 50 features of Class enc. are in figure 4.4.

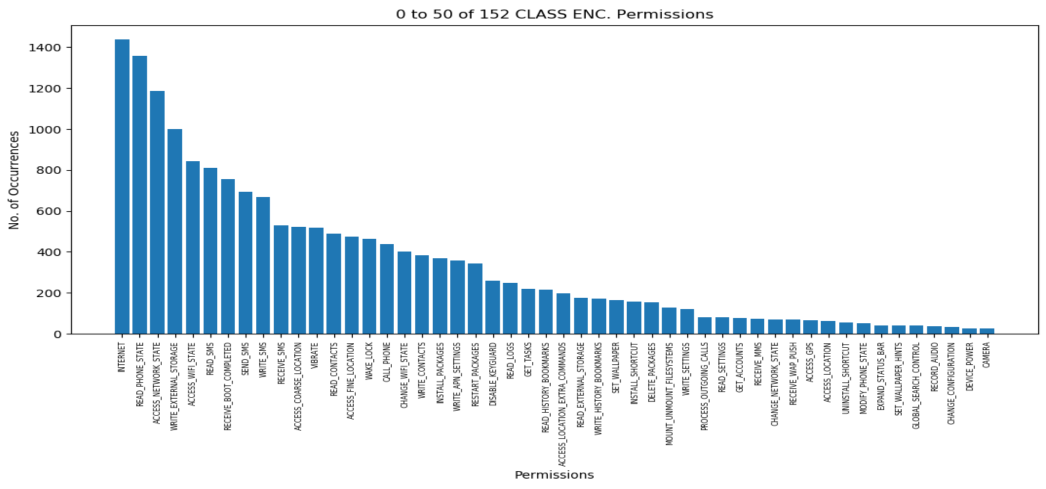


Figure 4.4: Permission Found from Class Enc.

PR1	PR2	PR3	-	PR4	PR5	PR6
1	1	0	-	1	1	1

Table 4.9: A Permission Pattern of Class Enc.

Family	PR1	PR2	PR3	-	PR4	PR5	PR6
DroidDream	1	1	1	-	1	0	1
DroidDream	1	1	0	-	1	0	1
GoldDream	1	0	0	-	1	0	0
GoldDream	1	1	0	-	1	0	1
YZHC	1	1	0	-	1	1	1
YZHC	1	1	0	-	1	1	1

Table 4.10: Family Permission Pattern of Class Enc.

4.1.5 Combination of Trivial and String Encryption

115 normal permissions including 18 requested permissions found from Trivial+String enc. Much more different from above four techniques. Above four techniques have 152 permissions but this combination techniques have only 115 permissions. Number of requested permissions with other techniques are same. Figure 4.5 shows top 50 features that are found from Trivial+String enc.

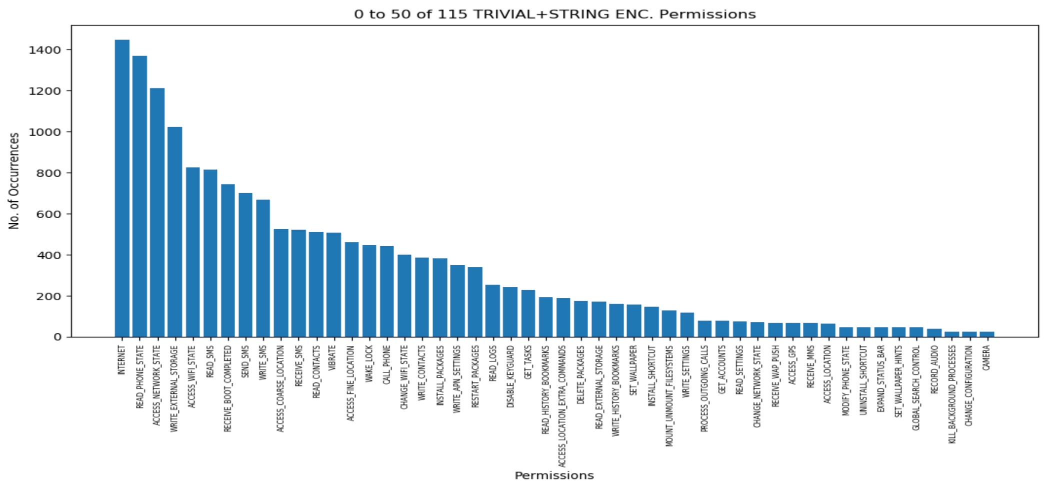


Figure 4.5: Permission Found from Trivial+String Enc.

PR1	PR2	PR3	-	PR4	PR5	PR6
1	1	0	-	1	1	1

Table 4.11: A Permission Pattern of Trivial+String Enc.

Family	PR1	PR2	PR3	-	PR4	PR5	PR6
FakePlayer	0	0	0	-	0	1	0
FakePlayer	0	0	0	-	0	1	0
Geinimi	1	1	1	-	1	1	1
Geinimi	1	1	1	-	0	1	1
Pjapps	1	1	1	-	1	1	1
Pjapps	1	1	1	-	1	1	1

Table 4.12: Family Permission Pattern of Trivial+String Enc.

4.1.6 Combination of Trivial, String and Reflection Encryption

Comparing to previous technique, number of permissions found from this technique is decreased to 111. Number of requested permissions is same as it was. Figure 4.6 is showing permissions found from this technique.

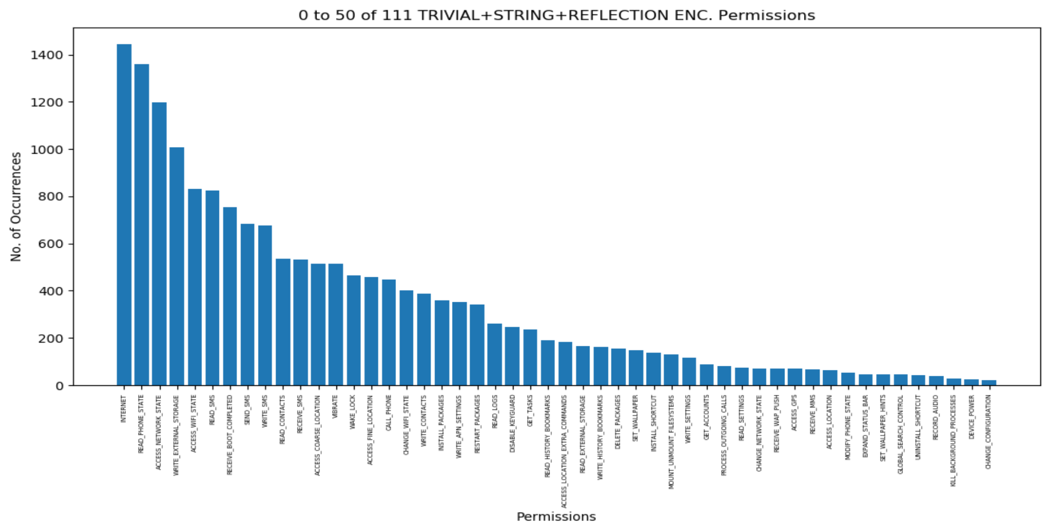


Figure 4.6: Permission Found from Trivial+String+Reflection Enc.

PR1	PR2	PR3	-	PR4	PR5	PR6
1	1	1	-	0	0	1

Table 4.13: A Permission Pattern of Trivial+String+Reflection Enc.

Family	PR1	PR2	PR3	-	PR4	PR5	PR6
DroidDreamLight	1	1	1	-	1	0	1
DroidDreamLight	1	1	1	-	1	0	1
DroidKungFu4	1	1	0	-	1	0	1
DroidKungFu4	1	1	1	-	1	0	1
AnserverBot	1	1	1	-	1	1	0
AnserverBot	1	1	1	-	1	1	0

Table 4.14: Family Permission Pattern of Trivial+String+Reflection Enc.

4.1.7 Combination of Trivial, String, Reflection and Class Encryption

With contrast of other techniques, most a smaller number of permissions found from this technique. Only 107 permissions found this combination. Number of requested permissions is same, 18. Figure 4.7 shows this technique most found features.

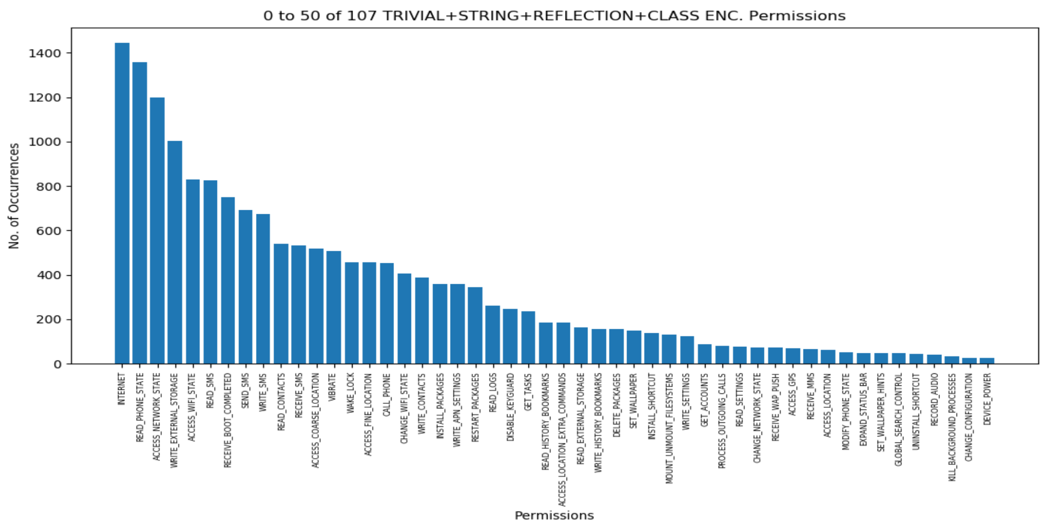


Figure 4.7: Permission Found from Trivial+String+Reflection+Class Enc.

PR1	PR2	PR3	-	PR4	PR5	PR6
1	0	1	-	1	0	1

Table 4.15: A Permission Pattern of Trivial+String+Reflection+Class Enc.

Family	PR1	PR2	PR3	-	PR4	PR5	PR6
GoldDream	1	1	0	-	1	1	1
GoldDream	1	1	0	-	1	1	1
Bgserv	1	1	0	-	1	1	1
Bgserv	1	1	0	-	1	1	1
jSMSHider	1	0	0	-	1	0	0
jSMSHider	1	1	1	-	1	0	0

Table 4.16: A Family Permission Pattern of Trivial+String+Reflection+Class Enc.

4.2 API CALL ANALYSIS

In this section this paper will illustrate every obfuscated techniques suspicious API call analysis report. Besides, every techniques API call pattern and technique wise a malware family pattern also given.

Full Version	Short Version
getInputStream	APC1
openConnection	APC2
getDeviceId	APC3
getPackageManager	APC4
getSubscriberId	APC5
getAssets	APC6

Table 4.17: Shortened Name for API Call

4.2.1 Trivial Encryption

From Trivial enc. analysis 23 suspicious API call found. They are in figure 4.8.

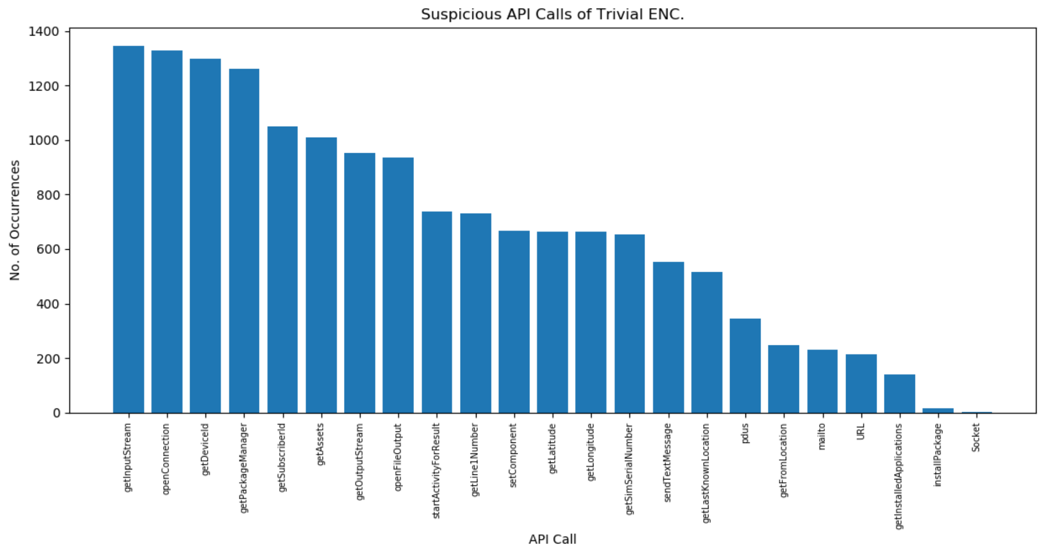


Figure 4.8: Suspicious API Call Found from Trivial Enc.

APC1	APC2	APC3	-	APC4	APC5	APC6
1	0	0	-	1	0	1

Table 4.18: An API Call Pattern of Trivial Enc.

Family	APC1	APC2	APC3	-	APC4	APC5	APC6
jSMSHider	1	1	1	-	1	1	0
jSMSHider	1	1	1	-	1	1	0
BaseBridge	1	1	0	-	1	1	1
BaseBridge	1	1	0	-	1	0	1
DroidKungFu1	1	1	1	-	1	0	1
DroidKungFu1	1	1	1	-	1	1	1

Table 4.19: Family API Call Pattern of Trivial Enc.

4.2.2 String Encryption

Like Trivial enc., there are same number of suspicious API call uses in String enc. They are given below -

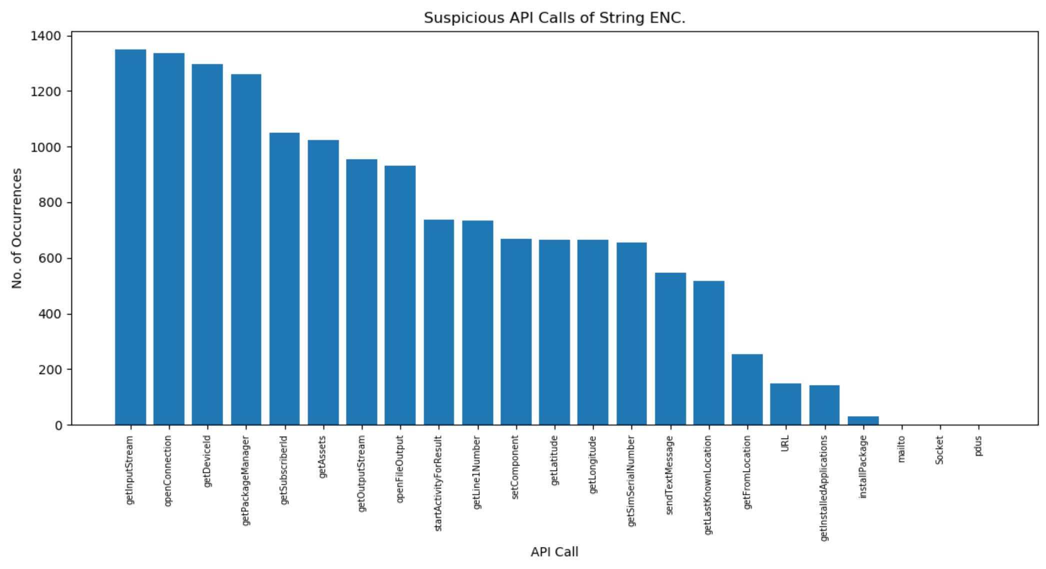


Figure 4.9: Suspicious API Call Found from String Enc.

APC1	APC2	APC3	-	APC4	APC5	APC6
1	1	1	-	1	0	1

Table 4.20: An API Call Pattern of String Enc.

Family	APC1	APC2	APC3	-	APC4	APC5	APC6
DroidKungFu1	1	1	1	-	1	1	1
DroidKungFu1	1	1	1	-	1	0	1
DroidDreamLight	1	1	1	-	1	1	1
DroidDreamLight	1	1	1	-	1	1	1
Pjapps	1	1	1	-	1	1	1
Pjapps	1	1	1	-	1	1	1

Table 4.21: Family API Call Pattern of String Enc.

4.2.3 Reflection Encryption

Similarly, with above two techniques, Reflection enc. have also same findings. Figure 4.10 shows usage of suspicious API call of Reflection enc.

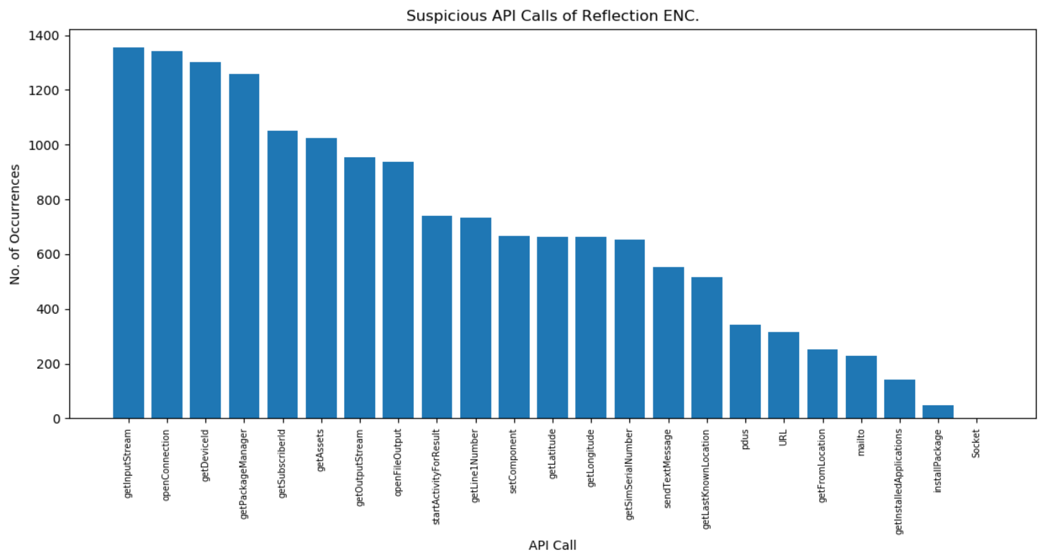


Figure 4.10: Suspicious API Call Found from Reflection Enc.

APC1	APC2	APC3	-	APC4	APC5	APC6
1	1	1	-	0	1	0

Table 4.22: An API Call Pattern of Reflection Enc.

Family	APC1	APC2	APC3	-	APC4	APC5	APC6
BaseBridge	1	1	0	-	1	1	1
BaseBridge	1	1	0	-	1	1	1
RogueSPPush	1	1	1	-	1	0	0
RogueSPPush	1	1	1	-	1	0	0
KMin	1	1	1	-	1	1	1
KMin	1	1	1	-	1	1	1

Table 4.23: Family API Call Pattern of Reflection Enc.

4.2.4 Class Encryption

From Class enc analysis, it shows that 20 suspicious API call used in this technique. It uses 3 less suspicious API call than other techniques. These are given in figure 4.11.

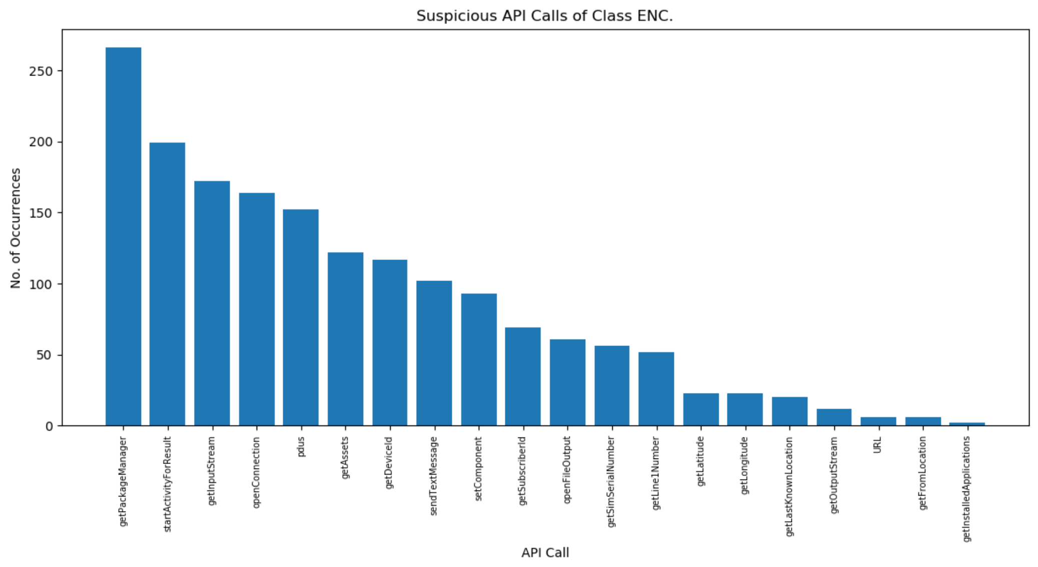


Figure 4.11: Suspicious API Call Found from Class Enc.

APC1	APC2	APC3	-	APC4	APC5	APC6
1	1	1	-	1	0	1

Table 4.24: An API Call Pattern of Class Enc.

Family	APC1	APC2	APC3	-	APC4	APC5	APC6
DroidDream	1	0	0	-	0	0	1
DroidDream	0	0	0	-	0	0	0
Zsone	0	0	0	-	1	0	1
Zsone	0	0	0	-	1	0	1
Asroot	0	0	0	-	0	0	1
Asroot	0	0	0	-	0	0	1

Table 4.25: Family API Call Pattern of Class Enc.

4.2.5 Combination of Trivial and String Encryption

Unlike with other techniques, 22 suspicious API call used in this technique. Figure 4.12 shows them.

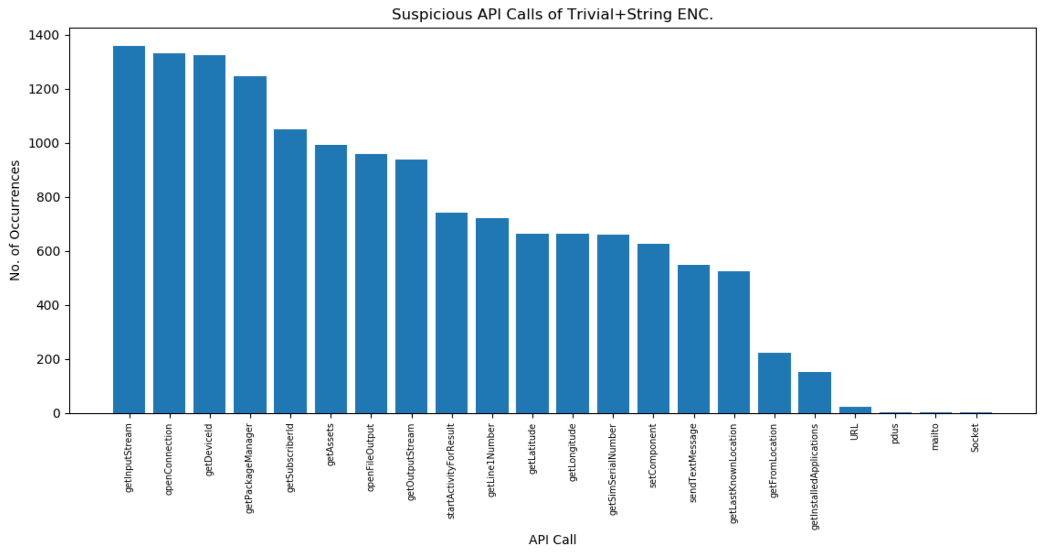


Figure 4.12: Suspicious API Call Found from Trivial+String Enc.

APC1	APC2	APC3	-	APC4	APC5	APC6
1	1	0	-	1	0	1

Table 4.26: An API Call Pattern of Trivial+String Enc.

Family	APC1	APC2	APC3	-	APC4	APC5	APC6
FakePlayer	0	0	0	-	0	0	0
FakePlayer	0	0	0	-	0	0	0
DroidKungFu4	1	1	1	-	1	1	1
DroidKungFu4	1	1	1	-	1	1	1
DroidDreamLight	1	1	1	-	1	1	1
DroidDreamLight	1	1	1	-	1	1	1

Table 4.27: Family API Call Pattern of Trivial+String Enc.

4.2.6 Combination of Trivial, String and Reflection Encryption

Like previous technique, same number of suspicious API call used in this technique. Usage are given in figure 4.13.

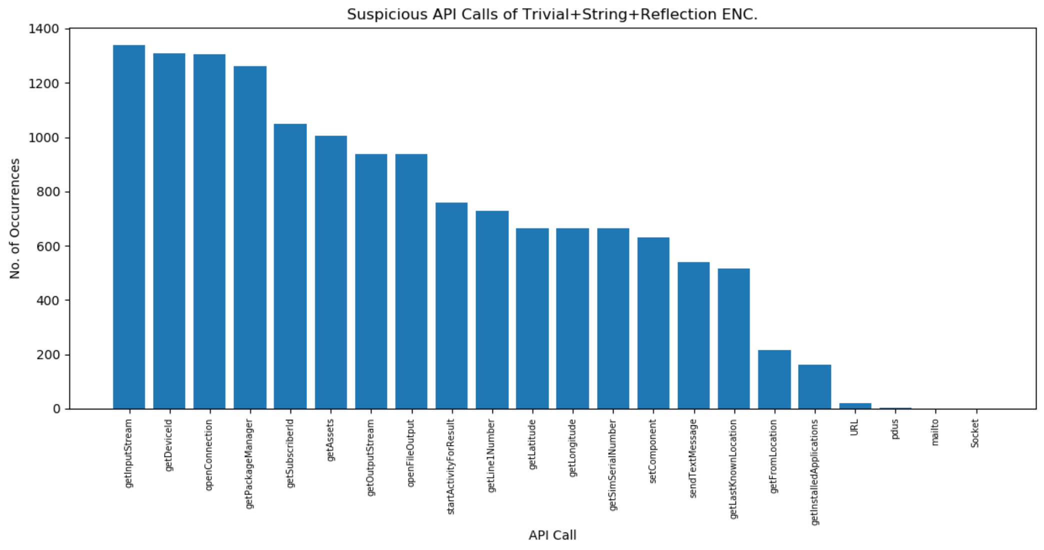


Figure 4.13: Suspicious API Call Found from Trivial+String+Reflection Enc.

APC1	APC2	APC3	-	APC4	APC5	APC6
1	1	1	-	1	1	0

Table 4.28: An API Call Pattern of Trivial+String+Reflection Enc.

Family	APC1	APC2	APC3	-	APC4	APC5	APC6
DroidDreamLight	1	1	1	-	1	1	1
DroidDreamLight	1	1	1	-	1	1	1
Geinimi	1	1	1	-	1	1	0
Geinimi	1	1	1	-	1	1	0
BaseBridge	1	1	1	-	1	1	1
BaseBridge	1	1	0	-	1	1	1

Table 4.29: Family API Call Pattern of Trivial+String+Reflection Enc.

4.2.7 Combination of Trivial, String, Reflection and Class Encryption

With contrast of other techniques, most a smaller number of permissions found from this technique. From trivial+string+reflection+class enc analysis this paper finds that 19 suspicious API call used in this technique. Figure 4.14 illustrated usage of suspicious API call of this technique.

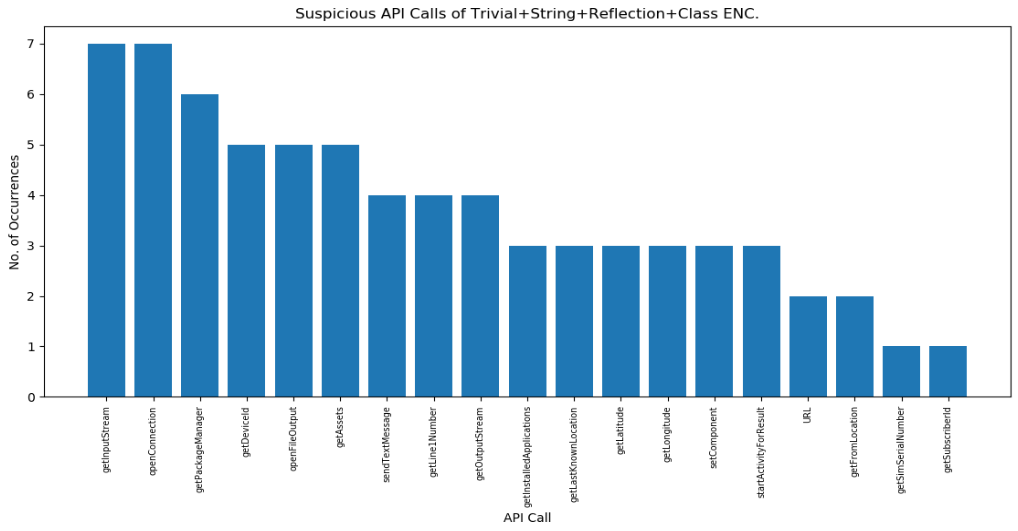


Figure 4.14: Suspicious API Call Found from Trivial+String+Reflection+Class Enc.

APC1	APC2	APC3	-	APC4	APC5	APC6
1	1	1	-	1	1	0

Table 4.30: An API Call Pattern of Trivial+String+Reflection+Class Enc.

Family	APC1	APC2	APC3	-	APC4	APC5	APC6
GoldDream	0	0	0	-	0	0	0
GoldDream	0	0	0	-	0	0	0
DroidKungFu3	0	0	0	-	0	0	0
DroidKungFu3	0	0	0	-	0	0	0
GingerMaster	0	0	0	-	0	0	0
GingerMaster	0	0	0	-	0	0	0

Table 4.31: Family API Call Pattern of Trivial+String+Reflection+Class Enc.

4.3 SYSTEM COMMAND ANALYSIS

In this section this paper will illustrate every obfuscated technique system command analysis report. Besides, every techniques System Command pattern and technique wise a malware family pattern also given.

Full Version	Short Version
mkdir	COM1
ln	COM2
su	COM3
getprop	COM4
ps	COM5
killall	COM6

Table 4.32: Shortened Name for System Command

4.3.1 Trivial Encryption

This analysis find that 12 system command used in this technique. They are given in figure 4.15

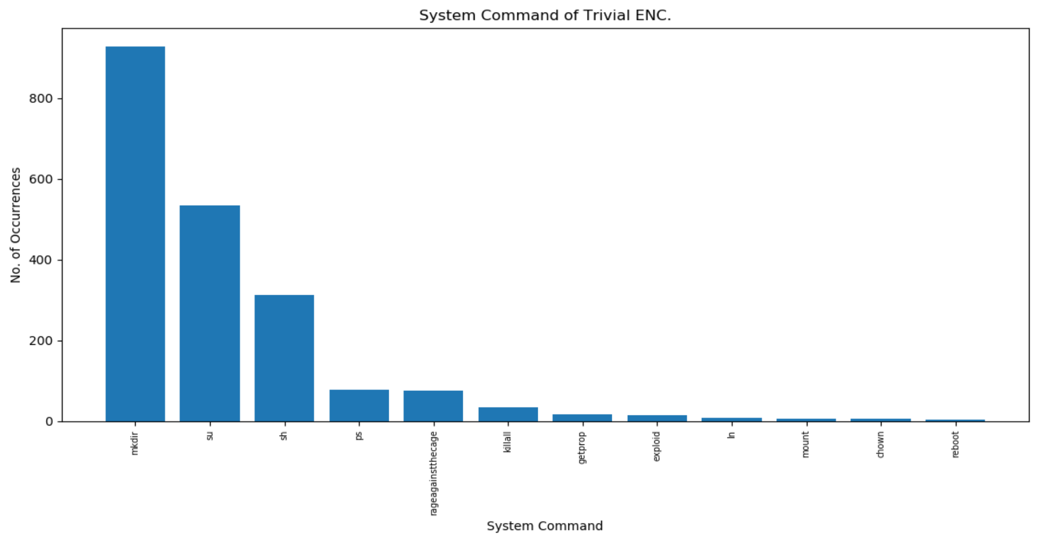


Figure 4.15: System Command Found from Trivial Enc.

COM1	COM2	COM3	-	COM4	COM5	COM6
1	1	1	-	0	1	0

Table 4.33: A System Command Pattern of Trivial Enc.

Family	COM1	COM 2	COM3	-	COM4	COM5	COM6
jSMSHider	0	0	1	-	0	0	0
jSMSHider	0	0	1	-	0	0	0
Zsone	1	0	1	-	0	0	0
Zsone	1	0	1	-	0	0	0
DroidKungFu2	1	0	1	-	0	0	0
DroidKungFu2	1	0	1	-	1	0	0

Table 4.34: Family System Command Pattern of Trivial Enc.

4.3.2 String Encryption

13 system command used in String enc. technique. All are showing in figure 4.16.

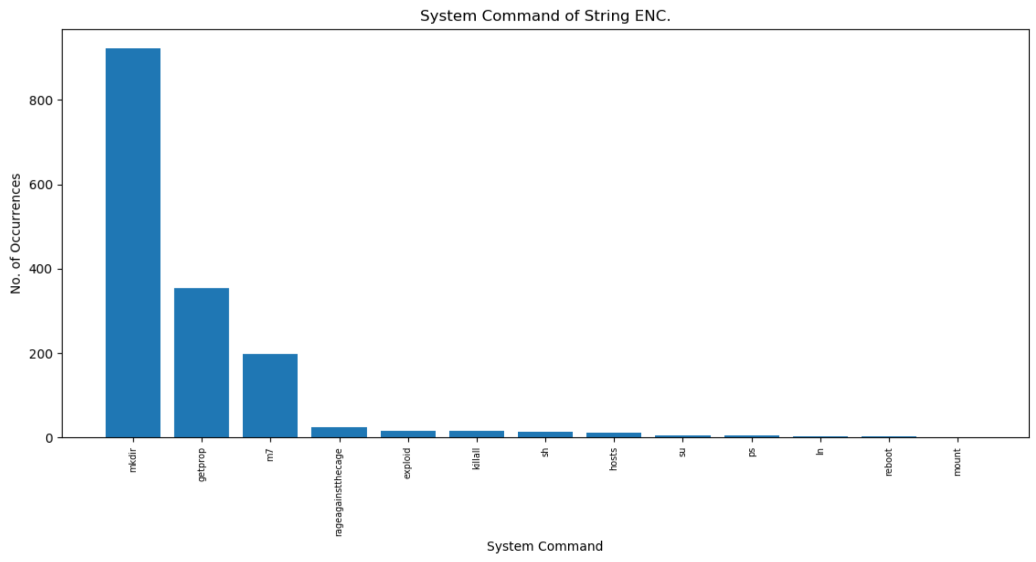


Figure 4.16: System Command Found from String Enc.

COM1	COM2	COM3	-	COM4	COM5	COM6
1	0	0	-	0	0	1

Table 4.35: A System Command Pattern of String Enc.

Family	COM1	COM2	COM3	-	COM4	COM5	COM6
DroidKungFu1	1	0	0	-	1	0	0
DroidKungFu1	1	0	0	-	1	0	0
Pjapps	1	0	1	-	0	0	0
Pjapps	1	0	0	-	0	0	0
AnserverBot	1	0	0	-	0	0	0
AnserverBot	1	0	0	-	0	0	0

Table 4.36: Family System Command Pattern of String Enc.

4.2.3 Reflection Encryption

From Reflection enc. analysis, 14 system command used in this technique. Comparing to first two techniques, number of suspicious API call use less in this technique. Figure 4.17 shows analysis result.

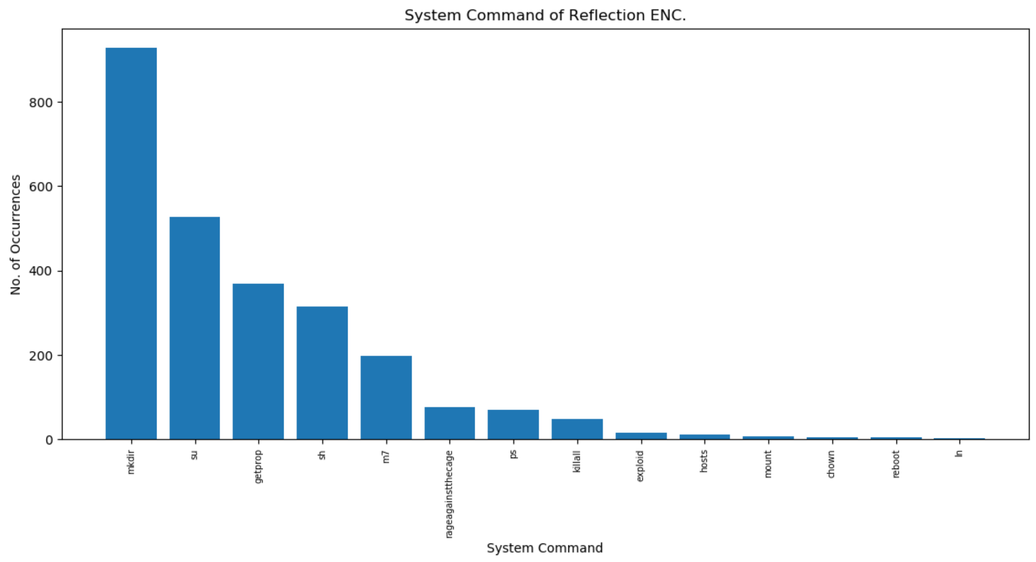


Figure 4.17: System Command Found from Reflection Enc.

COM1	COM2	COM3	-	COM4	COM5	COM6
1	0	1	-	0	0	0

Table 4.37: A System Command Pattern of Reflection Enc.

Family	COM1	COM2	COM3	-	COM4	COM5	COM6
BaseBridge	1	0	0	-	0	0	0
BaseBridge	1	0	0	-	0	0	0
KMin	1	0	1	-	0	0	0
KMin	1	0	1	-	0	0	0
jSMSHider	0	0	1	-	0	0	0
jSMSHider	1	0	1	-	0	0	0

Table 4.38: Family System Command Pattern of Reflection Enc.

4.3.4 Class Encryption

Class enc. uses 9 system command. That is much less than above three techniques. Figure 4.18 shows usage of suspicious API call of this technique.

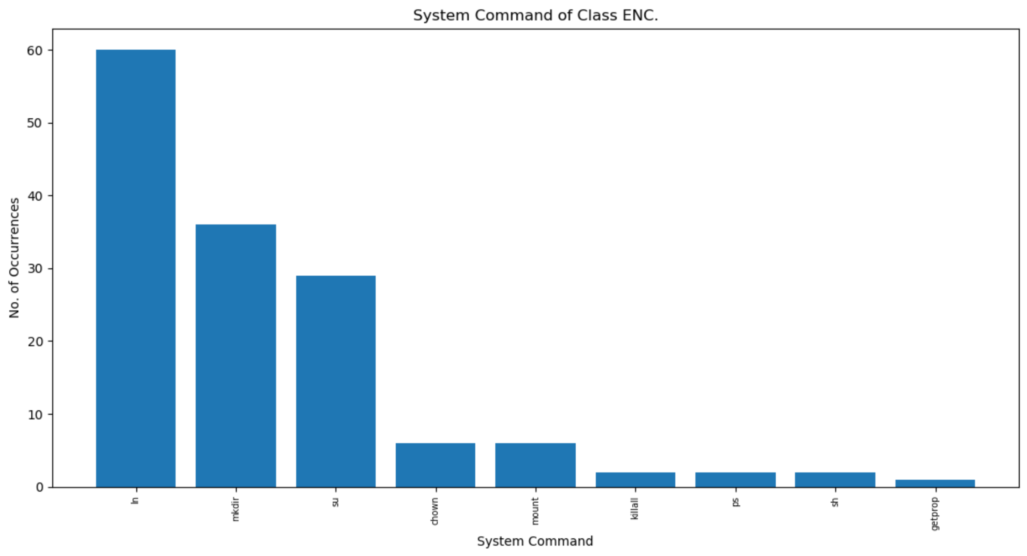


Figure 4.18: System Command Found from Class Enc.

COM1	COM2	COM3	-	COM4	COM5	COM6
1	0	1	-	0	0	0

Table 4.39: A System Command Pattern of Class Enc.

Family	COM1	COM2	COM3	-	COM4	COM5	COM6
DroidDream	0	0	0	-	0	0	1
DroidDream	0	0	0	-	0	0	0
DroidKungFu4	0	1	0	-	0	0	0
DroidKungFu4	0	1	0	-	0	0	0
BeanBot	0	0	1	-	0	0	0
BeanBot	0	0	1	-	0	0	0

Table 4.40: Family System Command Pattern of Class Enc.

4.3.5 Combination of Trivial and String Encryption

Similarity also shown in this technique with previous technique.10 system command used in this technique. They are demonstrating below in figure 4.19.

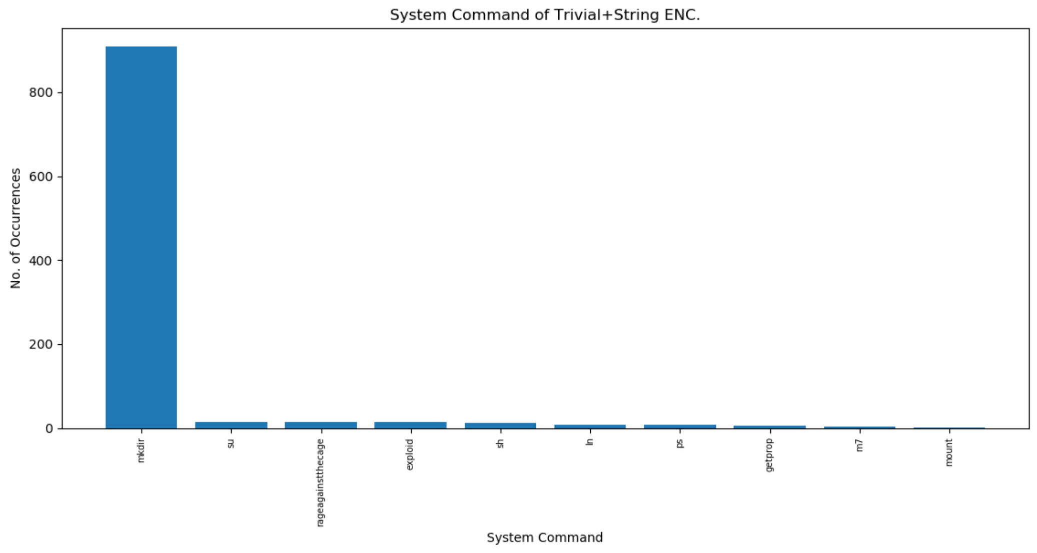


Figure 4.19: System Command Found from Trivial+String Enc.

COM1	COM2	COM3	COM4	COM5
1	0	0	0	0

Table 4.41: A System Command Pattern of Trivial+String Enc.

Family	COM1	COM2	COM3	-	COM4	COM5
FakePlayer	0	0	0	-	0	0
FakePlayer	0	0	0	-	0	0
GoldDream	1	0	0	-	0	0
GoldDream	1	0	0	-	0	0
DroidKungFu4	1	1	1	-	0	1
DroidKungFu4	1	1	1	-	0	1

Table 4.42: Family System Command Pattern of Trivial+String Enc.

4.3.6 Combination of Trivial, String and Reflection Encryption

This technique also not so much used system command as other techniques used. Only 8 system command used in this technique. Figure 4.20 shows 8 system commands.

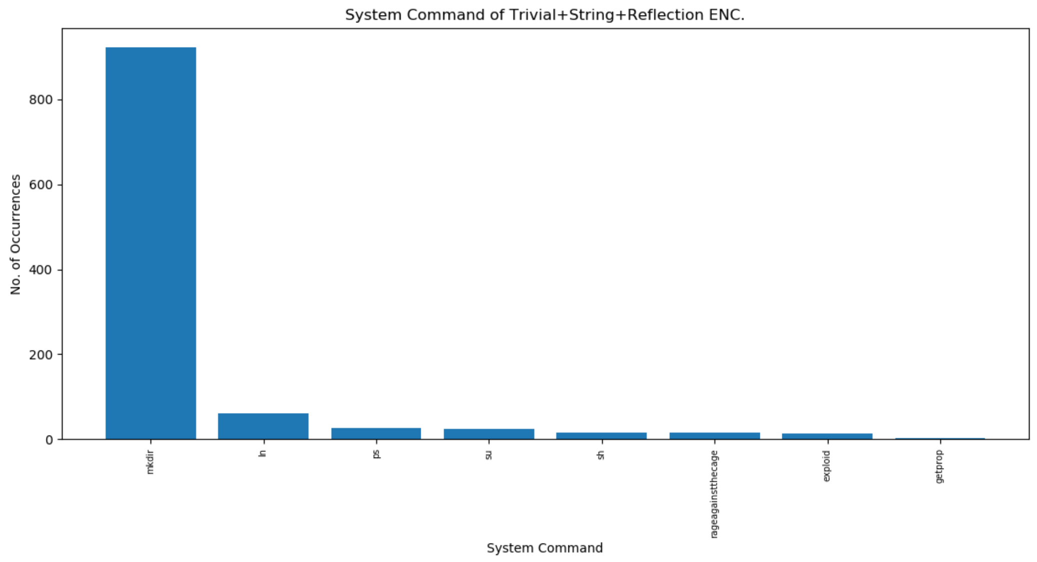


Figure 4.20: System Command Found from Trivial+String+Reflection Enc.

COM1	COM2	COM3	-	COM4	COM5
0	1	1	-	0	1

Table 4.43: A System Command Pattern of Trivial+String+Reflection Enc.

Family	COM1	COM2	COM3	-	COM4	COM5
DroidDreamLight	0	0	0	-	0	0
DroidDreamLight	0	0	0	-	0	0
DroidKungFu1	1	0	0	-	0	0
DroidKungFu1	1	0	0	-	0	0
Geinimi	0	1	1	-	0	1
Geinimi	1	1	0	-	0	0

Table 4.44: Family System Command Pattern of Trivial+String+Reflection Enc.

4.3.7 Combination of Trivial, String, Reflection and Class Encryption

A big difference found in this technique. Number of system command used in this technique is most less than other techniques. 4 system command used in this technique only. All are showing in figure 4.21.

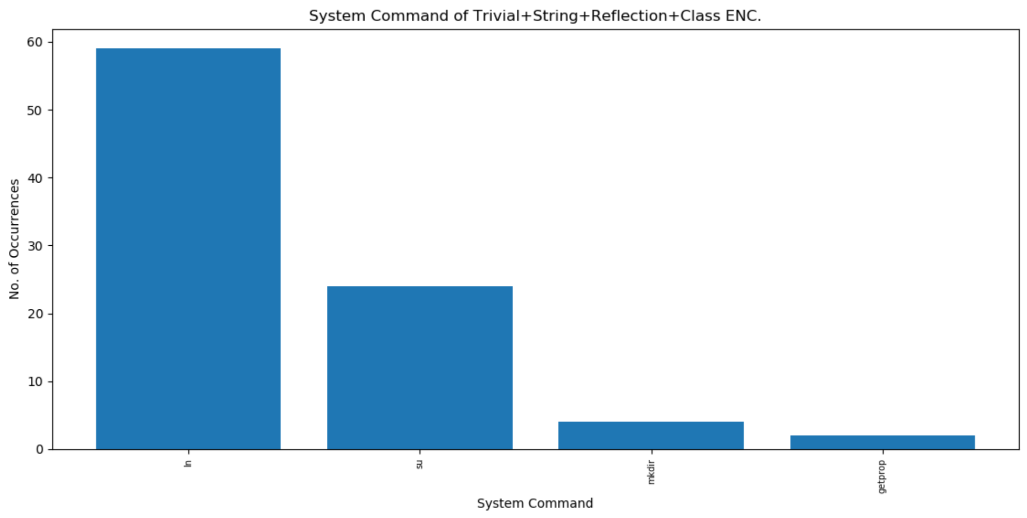


Figure 4.21: System Command Found from Trivial+String+Reflection+Class Enc.

COM1	COM2	COM3	COM4
0	1	0	0

Table 4.45: A System Command Pattern of Trivial+String+Reflection+Class Enc.

Family	COM1	COM2	COM3	COM4
GoldDream	0	0	0	0
GoldDream	0	0	0	0
DroidKungFu3	0	0	1	0
DroidKungFu3	0	0	1	0
ADRD	0	1	0	0
ADRD	0	1	0	0

Table 4.46: Family System Command Pattern of Trivial+String+Reflection+Class Enc.

4.4 INTENT FILTER ANALYSIS

In this section this paper will illustrate every obfuscated technique intent analysis report. Besides, every technique intent pattern and technique wise a malware family pattern also given.

Full Version	Short Version
BOOT_COMPLETED	IN1
CONTENT_CHANGED	IN2
PHONE_STATE	IN3
NEW_OUTGOING_CALL	IN4
SCREEN_ON	IN5
SCREEN_OFF	IN6

Table 4.47: Shortened Name for Intent

4.4.1 Trivial Encryption

Trivial enc. uses most number intent filter than other techniques. 90 intent filter used in this technique. Figure 4.22 shows top 50 intent filter used in this technique.

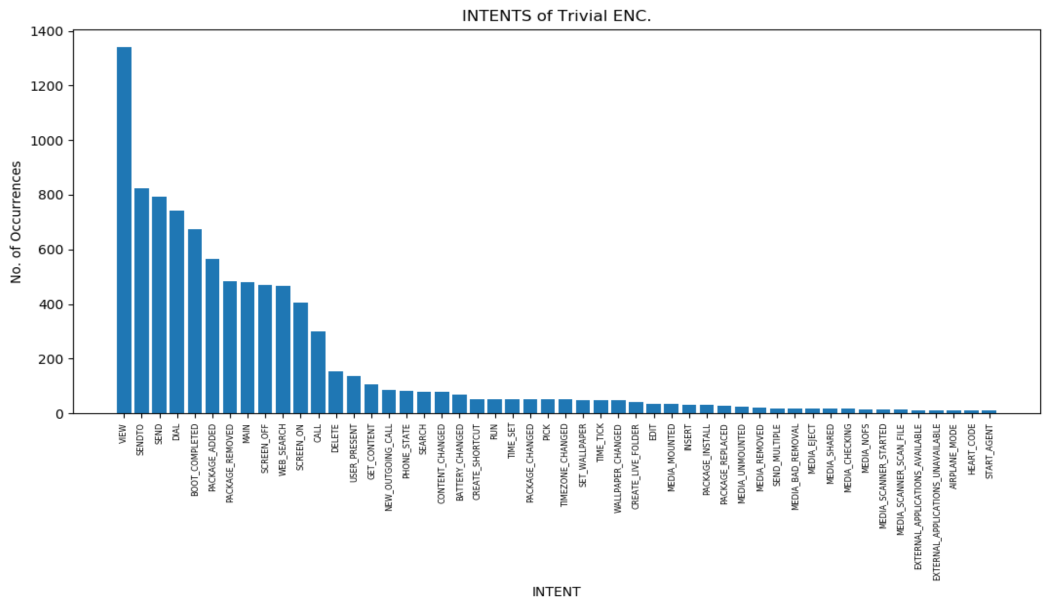


Figure 4.22: Intent Filter Found from Trivial Enc.

IN1	IN2	IN3	-	IN4	IN5	IN6
0	0	0	-	0	1	1

Table 4.48: An Intent Pattern of Trivial Enc.

Family	IN1	IN2	IN3	-	IN4	IN5	IN6
jSMSHider	0	0	0	-	0	0	0
jSMSHider	0	0	0	-	0	0	0
GoldDream	1	0	0	-	1	0	0
GoldDream	1	0	0	-	1	0	0
DroidDreamLight	1	0	1	-	0	0	1
DroidDreamLight	1	0	1	-	0	0	1

Table 4.49: Family Intent Pattern of Trivial Enc.

4.4.2 String Encryption

String enc. uses 34 intent filters which is almost one third of Trivial enc. All have been shown in figure 4.23.

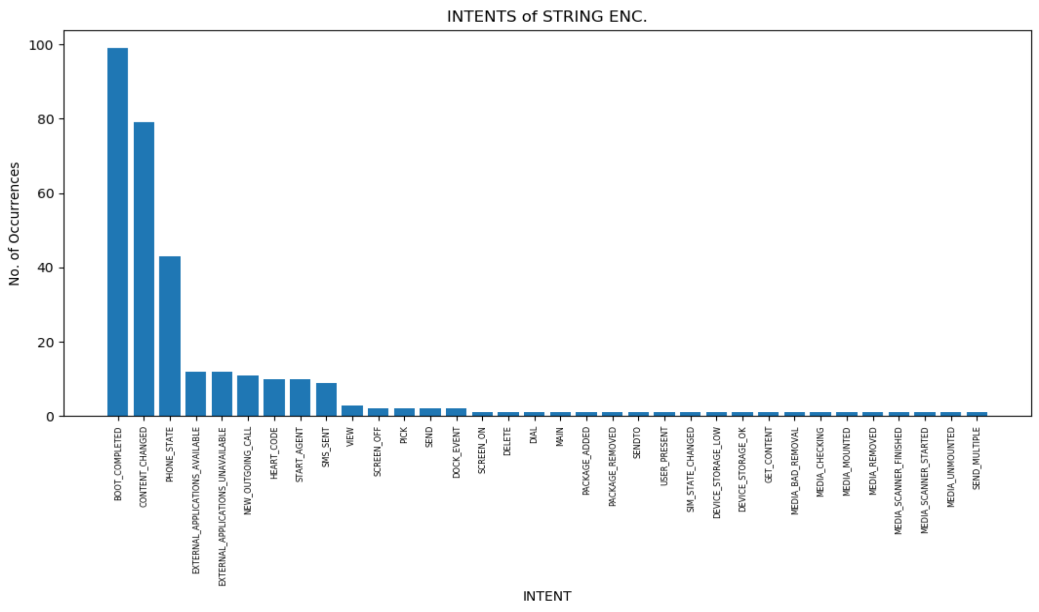


Figure 4.23: Intent Filter Found from String Enc.

IN1	IN2	IN3	-	IN4	IN5	IN6
1	0	1	-	1	0	0

Table 4.50: An Intent Pattern of String Enc.

Family	IN1	IN2	IN3	-	IN4	IN5	IN6
DroidKungFu1	0	0	0	-	0	0	0
DroidKungFu1	0	0	0	-	0	0	0
DroidKungFu4	0	1	0	-	0	0	0
DroidKungFu4	0	1	0	-	0	0	0
ADRD	1	0	0	-	0	0	0
ADRD	1	0	0	-	0	0	0

Table 4.51: Family Intent Pattern of String Enc.

4.4.3 Reflection Encryption

Like Trivial enc. this technique also used the greatest number of intent filters. 90 intent filters used in this technique. Top 50 usage given in figure 4.24.

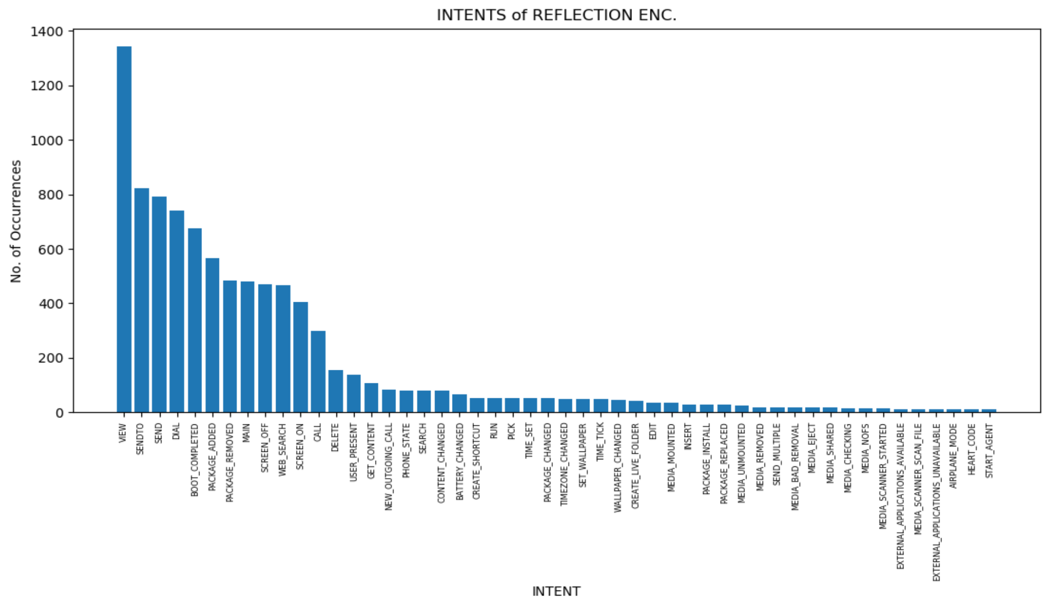


Figure 4.24: Intent Filter Found from Reflection Enc.

IN1	IN2	IN3	-	IN4	IN5	IN6
1	0	1	-	0	1	1

Table 4.52: An Intent Pattern of Reflection Enc.

Family	IN1	IN2	IN3	-	IN4	IN5	IN6
BaseBridge	0	0	0	-	0	1	1
BaseBridge	0	0	0	-	0	1	1
SndApps	1	0	0	-	0	0	0
SndApps	1	0	0	-	0	0	0
DroidKungFu2	1	0	0	-	0	0	0
DroidKungFu2	1	0	0	-	0	0	0

Table 4.53: Family Intent Pattern of Reflection Enc.

4.4.4 Class Encryption

68 intent filter used in Class enc. technique. Literally its used intent filter double of String enc. Top 50 used intent filter is in figure 4.25.

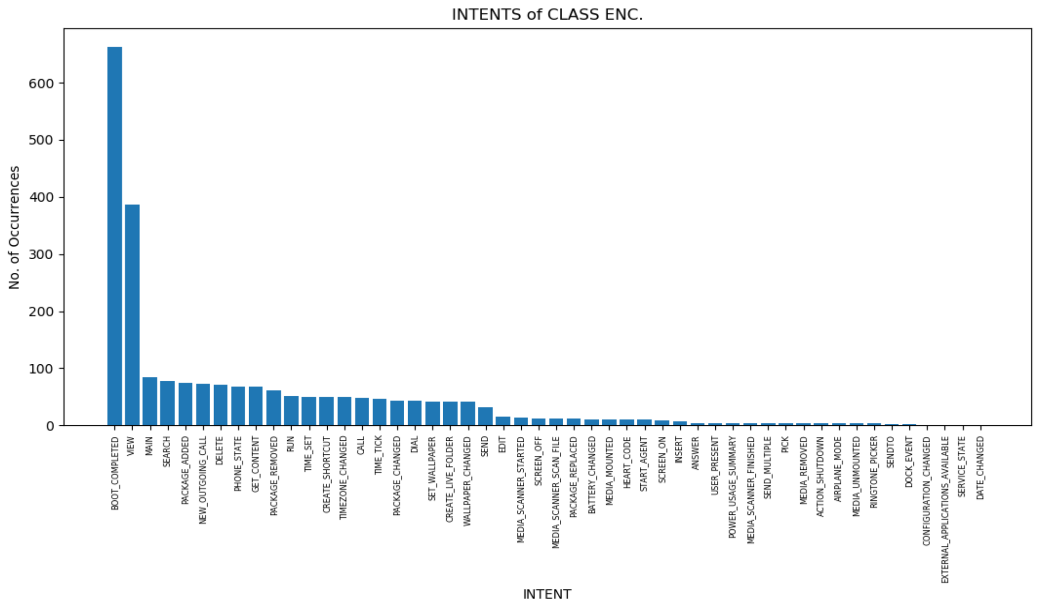


Figure 4.25: Intent Filter Found from Class Enc.

IN1	IN3	IN4	-	IN5	IN6
1	1	0	-	1	1

Table 4.54: An Intent Pattern of Class Enc.

Family	IN1	IN3	IN4	-	IN5	IN6
DroidDream	1	1	0	-	0	0
DroidDream	1	0	0	-	0	0
zHash	1	0	0	-	0	0
zHash	1	0	0	-	0	0
DroidKungFu3	1	0	0	-	0	1
DroidKungFu3	1	0	0	-	0	0

Table 4.55: Family Intent Pattern of Class Enc.

4.4.5 Combination of Trivial and String Encryption

This technique used much less intent filter than other single obfuscation techniques. Only 21 intent filter used in this technique. Figure 4.26 shows them all.

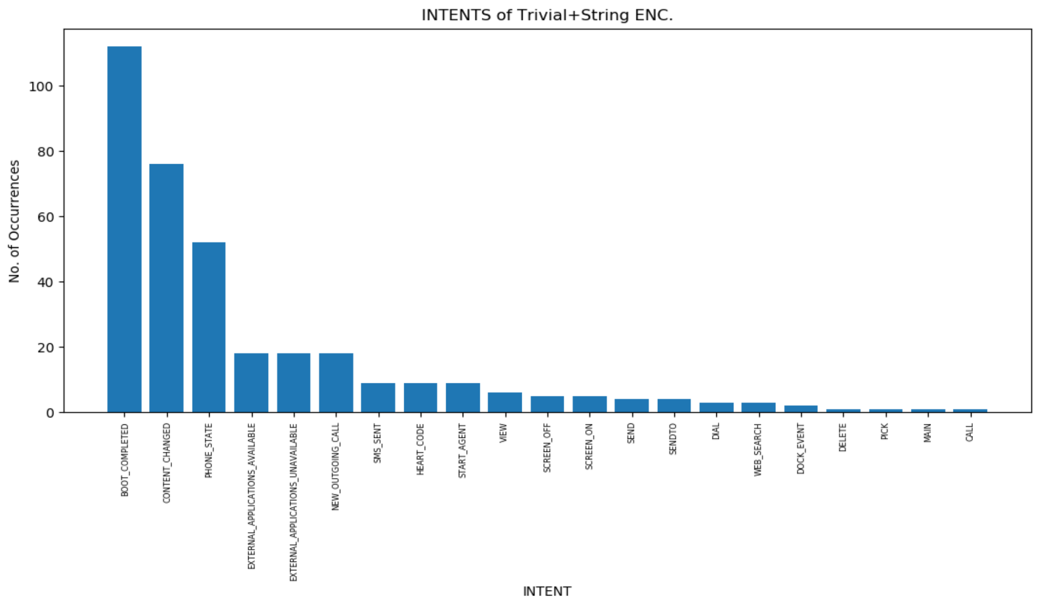


Figure 4.26: Intent Filter Found from Trivial+String Enc.

IN1	IN2	IN3	-	IN4	IN5	IN6
0	0	1	-	1	1	1

Table 4.56: An Intent Pattern of Trivial+String Enc.

Family	IN1	IN2	IN3	-	IN4	IN5	IN6
FakePlayer	0	0	0	-	0	0	0
FakePlayer	0	0	0	-	0	0	0
DroidKungFu4	1	0	1	-	1	0	0
DroidKungFu4	1	1	0	-	0	1	1
DroidDreamLight	1	0	1	-	0	0	0
DroidDreamLight	1	0	1	-	0	0	0

Table 4.57: Family Intent Pattern of Trivial+String Enc.

4.4.6 Combination of Trivial, String and Reflection Encryption

Likewise, previous technique, this combination techniques also have use number of intent filters. All intents filters are showing below in figure 4.27.

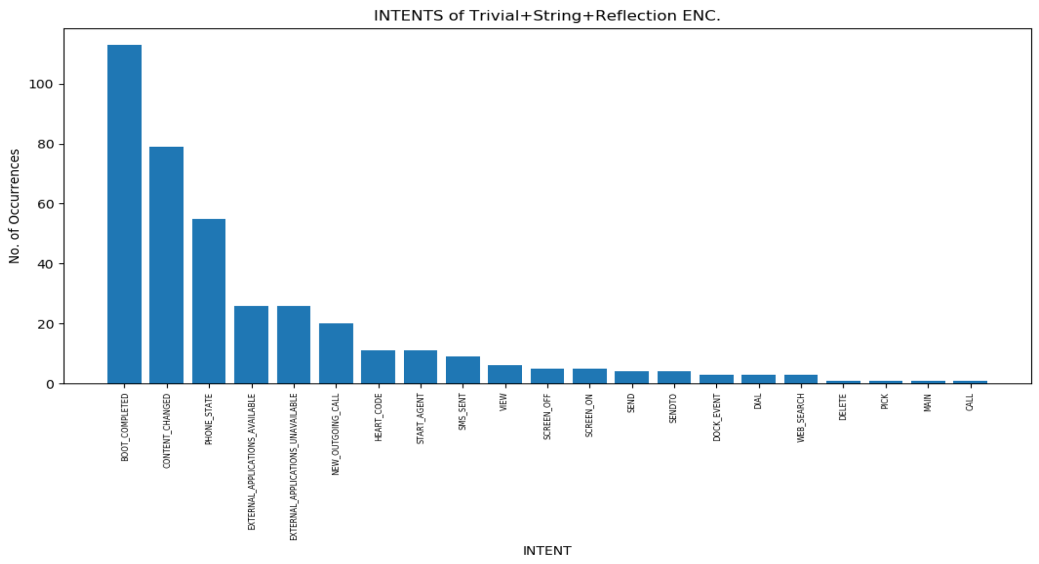


Figure 4.27: Intent Filter Found from Trivial+String+Reflection Enc.

IN1	IN2	IN3	-	IN4	IN5	IN6
0	0	1	-	1	1	1

Table 4.58: An Intent Pattern of Trivial+String+Reflection Enc.

Family	IN1	IN2	IN3	-	IN4	IN5	IN6
DroidDreamLight	1	0	1	-	0	0	0
DroidDreamLight	1	0	1	-	0	0	0
DroidKungFu4	0	1	0	-	0	0	0
DroidKungFu4	1	1	0	-	0	1	1
DroidKungFu3	1	0	1	-	1	0	0
DroidKungFu3	1	0	1	-	1	0	0

Table 4.59: Family Intent Pattern of Trivial+String+Reflection Enc.

4.4.7 Combination of Trivial, String, Reflection and Class Encryption

Unlikely this technique used most a smaller number of intent filter than other techniques. Only 3 intent used in this technique. Figure 4.28 illustrated them.

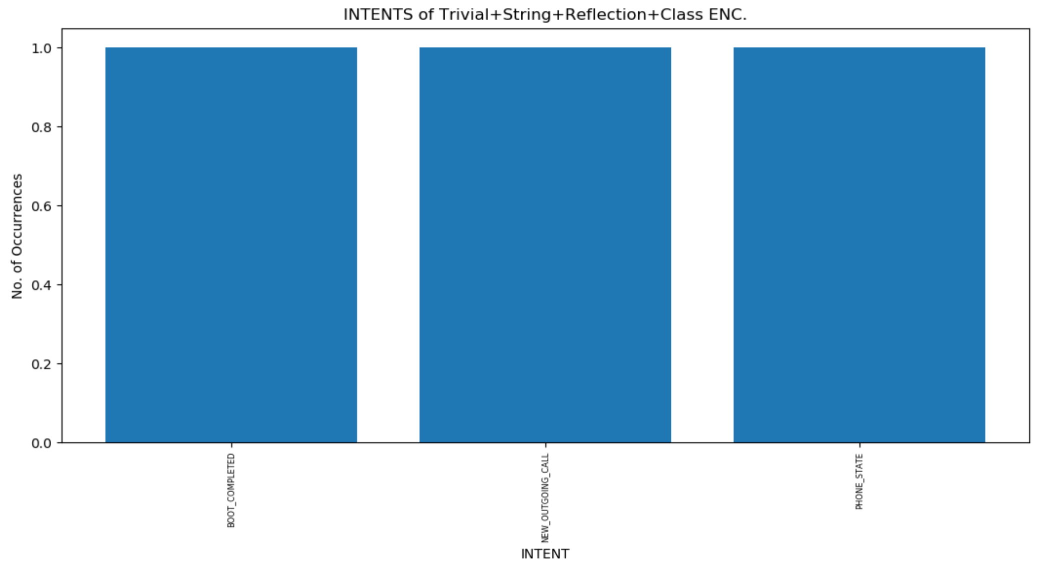


Figure 4.28: Intent Filter Found from Trivial+String+Reflection+Class Enc.

IN1	IN3	IN4
1	1	1

Table 4.60: An Intent Pattern of Trivial+String+Reflection+Class Enc.

Family	IN1	IN3	IN4
GoldDream	0	0	0
GoldDream	0	0	0
DroidKungFu3	1	1	1
DroidKungFu3	0	0	0
Geinimi	0	0	0
Geinimi	0	0	0

Table 4.61: Family Intent Pattern of Trivial+String+Reflection+Class Enc.

4.5 APP COMPONENT ANALYSIS

In this section this paper will illustrate every obfuscated technique app component analysis report. Besides, every techniques app component pattern and technique wise a malware family pattern also given.

Full Version	Short Version
Receiver	AC1
MainA	AC2
BaseABroadcastReceiver	AC3
MainActivity	AC4
BootReceiver	AC5
NotificationActivity	AC6

Table 4.62: Shortened Name for App Component

4.5.1 Trivial Encryption

Trivial obfuscation technique used 1774 app component. Top 50 usage of app component are in figure 4.29.

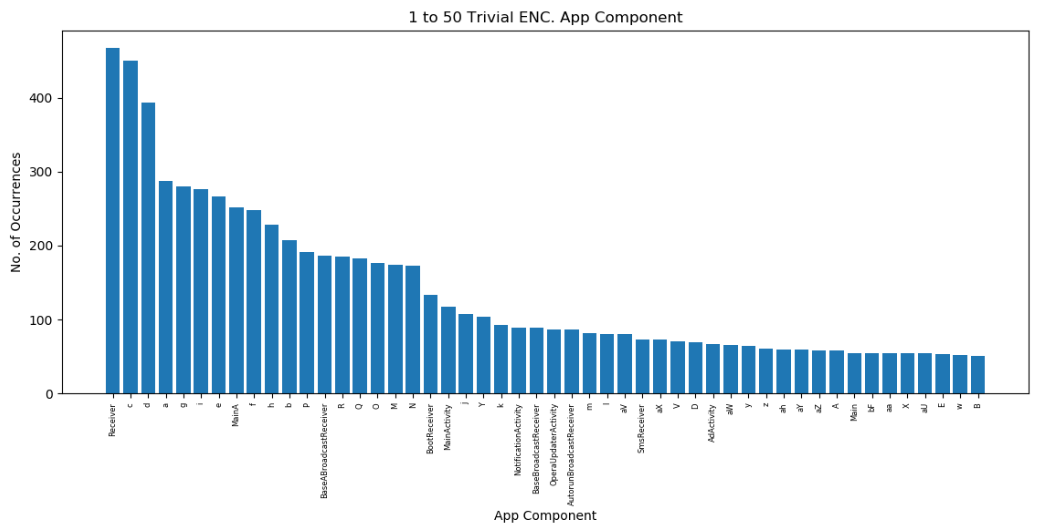


Figure 4.29: App Component Found from Trivial Enc.

AC1	AC2	AC3	-	AC4	AC5	AC6
0	0	0	-	0	1	0

Table 4.63: An App Component Pattern of Trivial Enc.

Family	AC1	AC2	AC3	-	AC4	AC5	AC6
jSMSHider	0	0	0	-	0	0	0
jSMSHider	0	0	0	-	0	0	0
AnserverBot	0	1	1	-	0	0	0
AnserverBot	0	1	1	-	0	0	0
ADRD	0	0	0	-	1	0	0
ADRD	0	0	0	-	1	0	0

Table 4.64: Family App Component Pattern of Trivial Enc.

4.5.2 String Encryption

String enc. used second highest number of app component among other techniques. 2381 app component used in this technique. Top 50 are in figure 4.30.

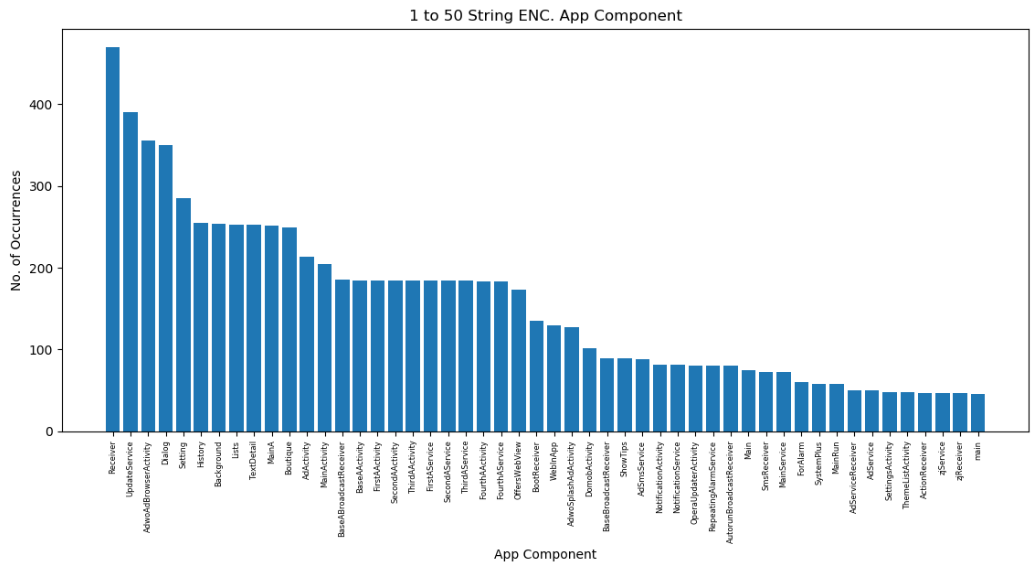


Figure 4.30: App Component Found from String Enc.

AC1	AC2	AC3	-	AC4	AC5	AC6
0	0	0	-	0	0	1

Table 4.65: An App Component Pattern of String Enc.

Family	AC1	AC2	AC3	-	AC4	AC5	AC6
DroidKungFu1	1	0	0	-	0	0	0
DroidKungFu1	1	0	0	-	0	0	0
Bgserv	0	0	0	-	0	1	0
Bgserv	0	0	0	-	0	1	0
KMin	0	0	0	-	1	1	0
KMin	0	0	0	-	1	0	0

Table 4.66: Family App Component Pattern of String Enc.

4.5.3 Reflection Encryption

Highest number of app component have been used by Reflection enc. 2429 app component applied in this technique. Top 50 apply of app component shown in figure 4.31.

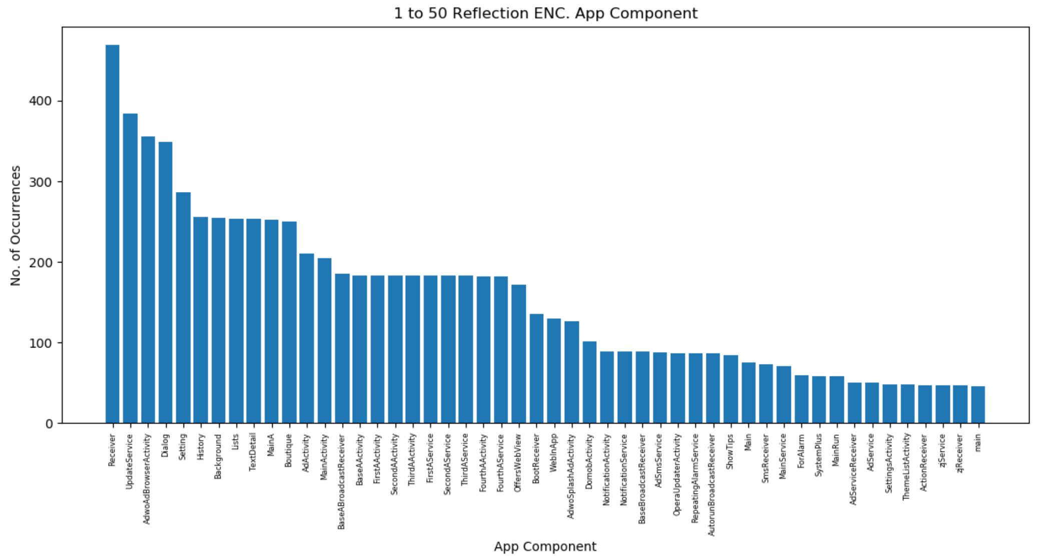


Figure 4.31: App Component Found from Reflection Enc.

AC1	AC2	AC3	-	AC4	AC5	AC6
0	0	0	-	0	0	1

Table 4.67: An App Component Pattern of Reflection Enc.

Family	AC1	AC2	AC3	-	AC4	AC5	AC6
BaseBridge	0	1	0	-	0	0	0
BaseBridge	0	0	0	-	0	0	0
AnserverBot	0	1	1	-	0	0	0
AnserverBot	0	1	1	-	0	0	0
DroidKungFu3	1	0	0	-	1	0	0
DroidKungFu3	1	0	0	-	0	0	0

Table 4.68: Family App Component Pattern of Reflection Enc.

4.5.4 Class Encryption

Lowest number of app component have been seen in Class enc. 68 app component utilized here. Figure 4.32 shows 50 most usage feature of app component.

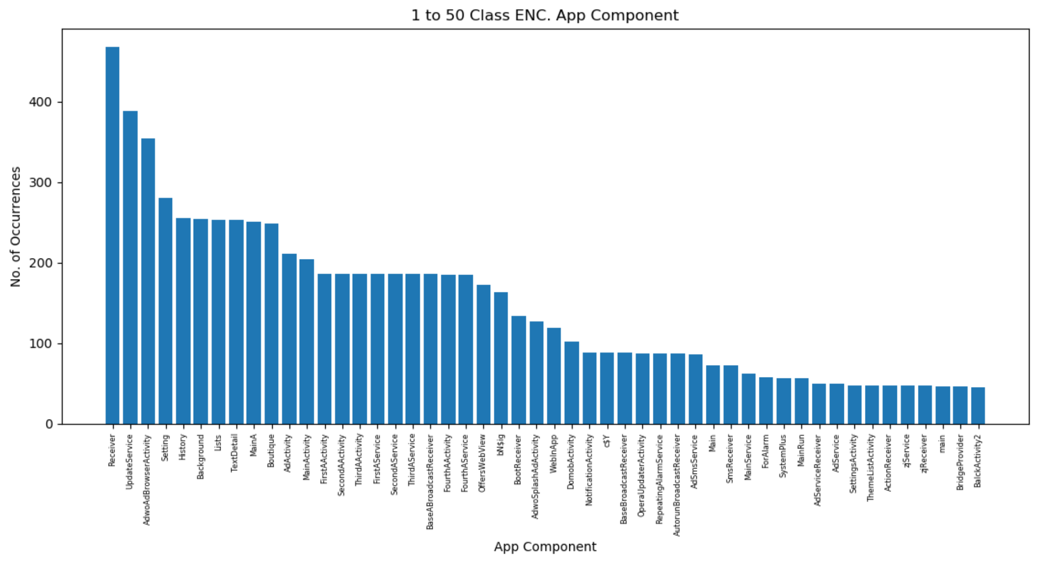


Figure 4.32: App Component Found from Class Enc.

AC1	AC2	AC3	-	AC4	AC5	AC6
0	0	0	-	1	0	0

Table 4.69: An App Component Pattern of Class Enc.

Family	AC1	AC2	AC3	-	AC4	AC5	AC6
DroidDream	0	0	0	-	0	0	0
DroidDream	0	0	0	-	0	0	0
RogueSPPush	0	0	0	-	1	0	0
RogueSPPush	0	0	0	-	1	0	0
YZHC	0	0	0	-	0	1	0
YZHC	0	0	0	-	0	1	0

Table 4.70: Family App Component Pattern of Class Enc.

4.5.5 Combination of Trivial and String Encryption

1579 app component have been employed in this combination technique. Most usage features are shown in figure 4.33.

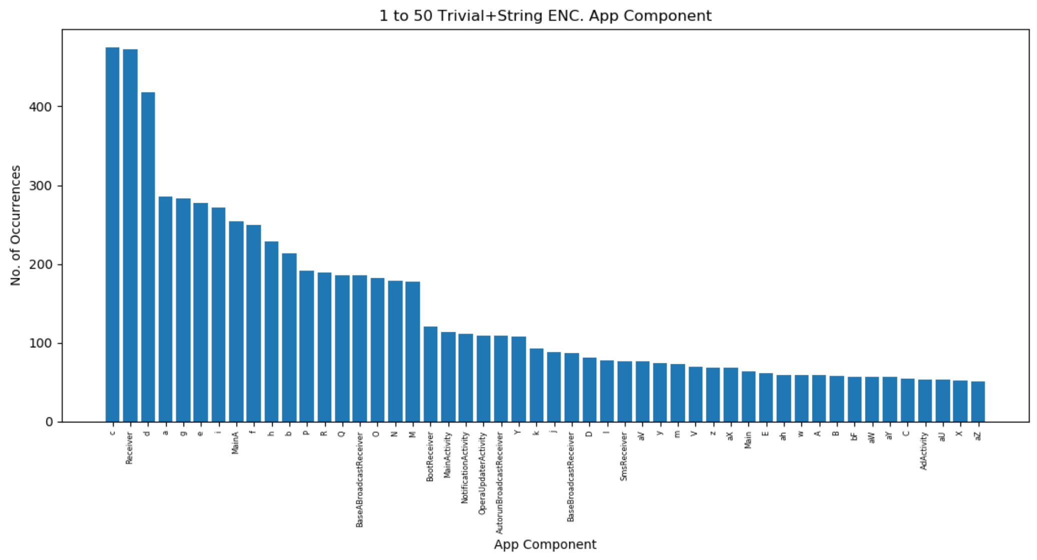


Figure 4.33: App Component Found from Trivial+String Enc.

AC1	AC2	AC3	-	AC4	AC5	AC6
0	0	0	-	0	1	0

Table 4.71: An App Component Pattern of Trivial+String Enc.

Family	AC1	AC2	AC3	-	AC4	AC5	AC6
FakePlayer	0	0	0	-	0	0	0
FakePlayer	0	0	0	-	0	0	0
zHash	0	0	0	-	1	0	0
zHash	0	0	0	-	1	0	0
AnserverBot	0	1	1	-	0	0	0
AnserverBot	0	1	1	-	0	0	0

Table 4.72: Family App Component Pattern of Trivial+String Enc.

4.5.6 Combination of Trivial, String and Reflection Encryption

1439 app component have been used by this complex obfuscation technique. Most usage features are shown in figure 4.34.

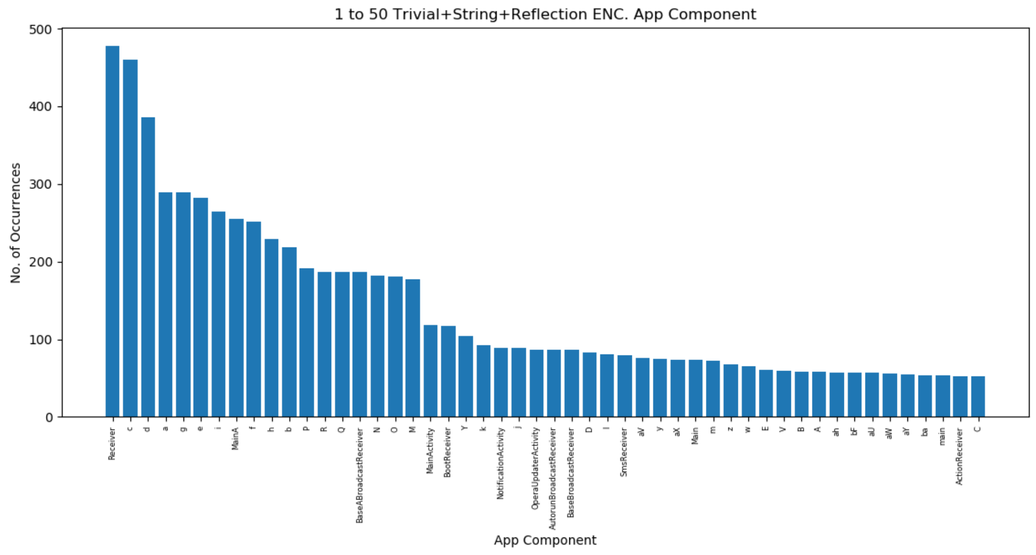


Figure 4.34: App Component Found from Trivial+String+Reflection Enc.

AC1	AC2	AC3	-	AC4	AC5	AC6
1	0	0	-	0	0	0

Table 4.73: An App Component Pattern of Trivial+String+Reflection Enc.

Family	AC1	AC2	AC3	-	AC4	AC5	AC6
DroidDreamLight	1	0	0	-	0	0	0
DroidDreamLight	1	0	0	-	0	0	0
DroidKungFu3	1	0	0	-	1	0	0
DroidKungFu3	1	0	0	-	1	0	0
BeanBot	0	0	0	-	1	0	0
BeanBot	0	0	0	-	1	0	0

Table 4.74: Family App Component Pattern of Trivial+String+Reflection Enc.

4.5.7 Combination of Trivial, String, Reflection and Class Encryption

This complex obfuscation strategy used 1365 app component. Figure 4.35 shows most usage features of app component of this strategy.

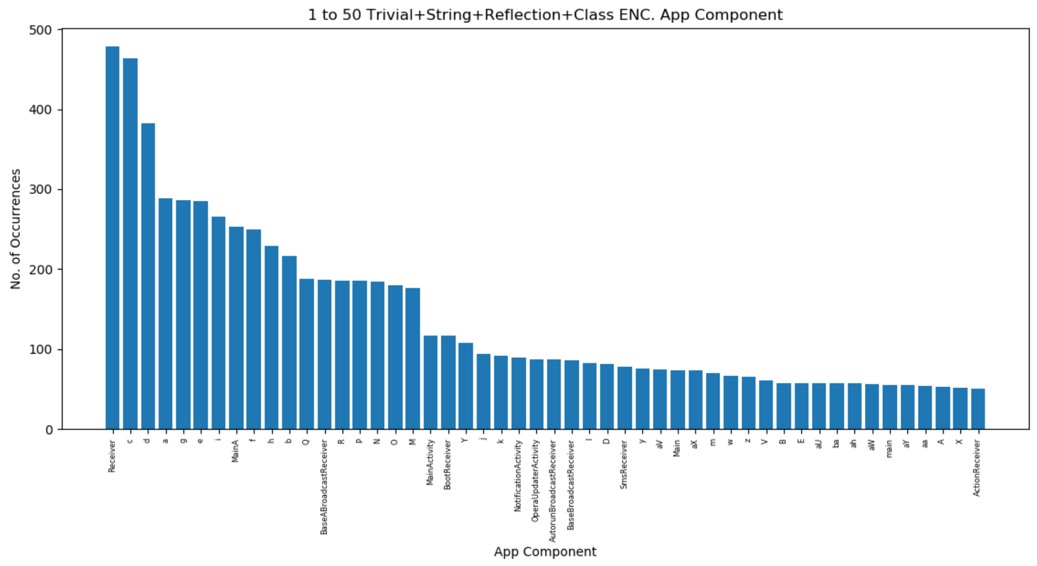


Figure 4.35: App Component Found from Trivial+String+Reflection+Class Enc.

AC1	AC2	AC3	-	AC4	AC5	AC6
0	0	0	-	0	0	1

Table 4.75: An App Component Pattern of Trivial+String+Reflection+Class Enc.

Family	AC1	AC2	AC3	-	AC4	AC5	AC6
GoldDream	0	0	0	-	0	0	0
GoldDream	0	0	0	-	0	0	0
DroidKungFu2	1	0	0	-	1	0	0
DroidKungFu2	1	0	0	-	1	0	0
NickySpy	0	0	0	-	0	1	0
NickySpy	0	0	0	-	1	1	0

Table 4.76: A Family App Component Pattern of Trivial+String+Reflection+Class Enc.

CHAPTER 5
CONCLUSIONS AND RECOMMENDATIONS

5.1 Findings and Contributions

In this study, AndroShow perform a static analysis of obfuscated Android malware applications. Permission, API call, Intent filter, App component and System call features are analyzed. AndroShow demonstrate confound malwares uses trend of these features**.** Several works have been done. Main contribution of this paper analysis is given below -

- Static analysis has been performed on obfuscated Android malware application.
- Analysis performs on five features - permission, API call, intent filter, app component, system call.
- Features pattern proposed in 2D matrix. Where column name is the feature tag name and rows are the 0/1 with family name.
- Most uses features demonstrate in 2D bar chart.
- Features extracted from obfuscated malware dataset, PRAGuard. This dataset contains 10,479 obfuscated malware applications with seven different obfuscation techniques.

5.2 Recommendations for Future Works

.

Future work will be classifying every muddle malware family wise. Detection of new malware app by machine learning based on features pattern can be a good thought.

REFERENCES

Aafer, Y., Du, W., & Yin, H. (2013, September). Droidapiminer: Mining API-level features for robust malware detection in android. In International conference on security and privacy in communication systems (pp. 86-103). Springer, Cham

Alasmary, H., Khormali, A., Anwar, A., Park, J., Choi, J., Nyang, D., & Mohaisen, A. (2019, February). Poster: Analyzing, Comparing, and Detecting Emerging Malware: A Graph-based Approach.

Android Developers. (n.d.-a). Permissions overview. Retrieved from https://developer.android.com/guide/topics/permissions/overview

Android Developers. (n.d.-b). Intents and Intent Filters. Retrieved from https://developer.android.com/guide/components/intents-filters

Android Developers. (n.d.-c). App components. Retrieved from https://developer.android.com/guide/components/fundamentals#Components

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. E. R. T. (2014, February). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Ndss* (Vol. 14, pp. 23-26).

Arslan, R. S., Doğru, İ. A., & Barişçi, N. (2019). Permission-Based Malware Detection System for Android Using Machine Learning Techniques. International Journal of Software Engineering and Knowledge Engineering, 29(01), 43-61.

Aung, Z., & Zaw, W. (2013). Permission-based android malware detection. International Journal of Scientific & Technology Research, 2(3), 228-234.

Ben, P. (2017, May 17). Google announces over 2 billion monthly active devices on Android [web log post]. Retrieved from https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users

Chang, S. C., Sun, Y. S., Chuang, W. L., Chen, M. C., Sun, B., & Takahashi, T. (2018, December). ANTSdroid: Using RasMMA Algorithm to Generate Malware Behavior Characteristics of Android Malware Family. In 2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC) (pp. 257-262). IEEE.

Da, C., Hongmei, Z., & Xiangli, Z. (2016, October). Detection of Android malware security on system calls. In 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC) (pp. 974-978). IEEE.

Dighe, S., Jondhale, A., & Salunke, D. J. Permission Based Android Malware Detection

Dimjaševic, M., Atzeni, S., Ugrina, I., & Rakamaric, Z. (2015). Android malware detection based on system calls. University of Utah, Tech. Rep.

Dong, Y. (2017). Android Malware Prediction by Permission Analysis and Data Mining.

Elish, K. O., Yao, D., & Ryder, B. G. (2015, May). On the need of precise inter-app ICC classification for detecting Android malware collusions. In Proceedings of IEEE mobile security technologies (MoST), in conjunction with the IEEE symposium on security and privacy.

Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). Androdialysis: Analysis of android intent effectiveness in malware detection. computers & security, 65, 121-134.

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011, October). Android permissions demystified. In Proceedings of the 18th ACM conference on Computer and communications security (pp. 627-638). ACM.

Feng, Y., Anand, S., Dillig, I., & Aiken, A. (2014, November). 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). ACM.

Fereidooni, H., Moonsamy, V., Conti, M., & Batina, L. (2016). Efficient classification of android malware in the wild using robust static features. Protecting Mobile Networks and Devices: Challenges and Solutions, 1, 181-209.

Firdaus, A., & Anuar, N. B. (2015). Root-exploit malware detection using static analysis and machine learning. In Proceedings of the fourth international conference on Computer Science & Computational Mathematics (ICCSCM 2015). Langkawi, Malaysia (pp. 177-183).

Ghani, S. M. A., Abdollah, M. F., Yusof, R., & Mas'ud, M. Z. (2015). Recognizing API Features for Malware Detection Using Static Analysis. Journal of Wireless Networking and Communications, 5(2A), 6-12.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012, June). Riskranker: scalable and accurate zero-day android malware detection. In Proceedings of the 10th international conference on Mobile systems, applications, and services (pp. 281-294). ACM.

He, N., Wang, T., Chen, P., Yan, H., & Jin, Z. (2018, December). An Android Malware Detection Method Based on Deep AutoEncoder. In Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference (pp. 88-93). ACM.

Hu, Y., Wang, H., Li, L., Guo, Y., Xu, G., & He, R. (2019, February). Want to Earn a Few Extra Bucks? A First Look at Money-Making Apps. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 332-343). IEEE.

Huang, C. Y., Tsai, Y. T., & Hsu, C. H. (2013). Performance evaluation on permission-based detection for android malware. In Advances in Intelligent Systems and Applications-Volume 2 (pp. 111-120). Springer, Berlin, Heidelberg.

IDC. (n.d.). Smartphone Market Share. Retrieved from https://www.idc.com/promo/smartphone-market-share/os

Iqbal, S., & Zulkernine, M. (2018, October). SpyDroid: A Framework for Employing Multiple Real-Time Malware Detectors on Android. In 2018 13th International Conference on Malicious and Unwanted Software (MALWARE) (pp. 1-8). IEEE

Jan, S. & Ondrej, D. (2018, July 25-27). Retrieved from https://www.rsaconference.com/writable/presentations/file_upload/tta-r09_android_malware_trends_on_a_global_scale_final.pdf

Jannatul, I. (2018, November 9). Pathao violates users' privacy. Retrieved from https://www.daily-sun.com/printversion/details/348905/2018/11/09/Pathao-violates-users%E2%80%99-privacy

Johnson, R., Wang, Z., Gagnon, C., & Stavrou, A. (2012, June). Analysis of android applications' permissions. In 2012 IEEE Sixth International Conference on Software Security and Reliability Companion (pp. 45-46). IEEE.

Kedziora, M., Gawin, P., Szczepanik, M., & Jozwiak, I. (2018). Android Malware Detection Using Machine Learning And Reverse Engineering. Computer Science & Information Technology (CS & IT). 95-107.

Kim, T., Kang, B., Rho, M., Sezer, S., & Im, E. G. (2019). A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. IEEE Transactions on Information Forensics and Security, 14(3), 773-788.

Kumar, R., Zhang, X., Khan, R. U., & Sharif, A. (2019). Research on Data Mining of Permission-Induced Risk for Android IoT Devices. Applied Sciences, 9(2), 277.

Li, C., Zhu, R., Niu, D., Mills, K., Zhang, H., & Kinawi, H. (2018). Android Malware Detection based on Factorization Machine. arXiv preprint arXiv:1805.11843.

Li, L., Bartel, A., Bissyandé, T. F., Klein, J., Le Traon, Y., Arzt, S., ... & McDaniel, P. (2015, May). Iccta: Detecting inter-component privacy leaks in android apps. In Proceedings of the 37th International Conference on Software Engineering-Volume 1 (pp. 280-291). IEEE Press.

Li, L., Bartel, A., Klein, J., & Le Traon, Y. (2014, September). Automatically exploiting potential component leaks in android applications. In 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (pp. 388-397). IEEE.

Maiorca, D., Ariu, D., Corona, I., Aresu, M., & Giacinto, G. (2015). Stealth attacks: An extended insight into the obfuscation effects on android malware. Computers & Security, 51, 16-31.

Malik, S., & Khatter, K. (2016). System call analysis of android malware families. Indian Journal of Science and Technology, 9(21).

Martín, I., Hernández, J. A., & de los Santos, S. (2019). Machine-Learning based analysis and classification of Android malware signatures. Future Generation Computer Systems.

Murtaz, M., Azwar, H., Ali, S. B., & Rehman, S. (2018, November). A framework for Android Malware detection and classification. In 2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS) (pp. 1-5). IEEE.

Parker, C., McDonald, J. T., Johnsten, T., & Benton, R. G. (2018, October). Android Malware Detection Using Step-Size Based Multi-layered Vector Space Models. In 2018 13th International Conference on Malicious and Unwanted Software (MALWARE) (pp. 1-10). IEEE.

Peiravian, N., & Zhu, X. (2013, November). Machine learning for android malware detection using permission and API calls. In 2013 IEEE 25th international conference on tools with artificial intelligence (pp. 300-305). IEEE.

Perry, E. (2017, December 7). What exactly IS an API? [web log post]. Retrieved from https://medium.com/@perrysetgo/what-exactly-is-an-API-69f36968a41f

Rana, M. S., Gudla, C., & Sung, A. H. (2018, December). Evaluating Machine Learning Models for Android Malware Detection: A Comparison Study. In Proceedings of the 2018 VII International Conference on Network, Communication and Computing (pp. 17-21). ACM.

Seo, S. H., Gupta, A., Sallam, A. M., Bertino, E., & Yim, K. (2014). Detecting mobile malware threats to homeland security through static analysis. Journal of Network and Computer Applications, 38, 43-53.

Shen, F., Del Vecchio, J., Mohaisen, A., Ko, S. Y., & Ziarek, L. (2018). Android malware detection using complex-flows. IEEE Transactions on Mobile Computing.

Shen, T., Zhongyang, Y., Xin, Z., Mao, B., & Huang, H. (2014, September). Detect Android malware variants using component based topology graph. In 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (pp. 406-413). IEEE.

Skovoroda, A., & Gamayunov, D. (2017, August). Automated static analysis and classification of Android malware using permission and API calls models. In 2017 15th Annual Conference on Privacy, Security and Trust (PST) (pp. 243-24309). IEEE.

Statista. (n.d.). Number of apps available in leading app stores as of 3rd quarter 2018. Retrieved from https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

Suarez-Tangil, G., Dash, S. K., Ahmadi, M., Kinder, J., Giacinto, G., & Cavallaro, L. (2017, March). DroidSieve: Fast and accurate classification of obfuscated android malware. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy (pp. 309-320). ACM.

Tchakounté, F., & Dayang, P. (2013). System calls analysis of malwares on android. International Journal of Science and Technology, 2(9), 669-674.

Tiwari, S. R., & Shukla, R. U. (2018, June). An Android Malware Detection Technique Using Optimized Permission and API with PCA. In 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 2611-2616). IEEE.

tutorialspoint. (n.d.). Android - Application Components. Retrieved from https://www.tutorialspoint.com/android/android_application_components.htm

Qamar, A., Karim, A., & Chang, V. (2019). Mobile malware attacks: Review, taxonomy & future directions. Future Generation Computer Systems.

Wahanggara, V., & Prayudi, Y. (2015, October). Malware detection through call system on android smartphone using vector machine method. In 2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic (CyberSec) (pp. 62-67). IEEE.

Wang, X., Zhang, D., Su, X., & Li, W. (2017). Mlifdect: android malware detection based on parallel machine learning and information fusion. Security and Communication Networks, 2017.

Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012, August). Droidmat: Android malware detection through manifest and API calls tracing. In 2012 Seventh Asia Joint Conference on Information Security (pp. 62-69). IEEE.

Xu, K., Li, Y., & Deng, R. H. (2016). Iccdetector: Icc-based malware detection on android. IEEE Transactions on Information Forensics and Security, 11(6), 1252-1264.

Yang, M., Wang, S., Ling, Z., Liu, Y., & Ni, Z. (2017). Detection of malicious behavior in android apps through API calls and permission uses analysis. Concurrency and Computation: Practice and Experience, 29(19), e4172.