2.18 Discussion

In this chapter, an extensive survey covering the fundamentals of malware, including its taxonomy, various detection approaches, and tools for malware analysis, is carried out. The classification of malware into different types, is explored, highlighting their unique characteristics and behaviors. Various malware detection techniques are discussed, including signature-based, and anomaly-based, emphasizing their strengths and limitations. The tools and frameworks commonly used in malware analysis, such as sandboxing, static and dynamic analysis tools, and reverse engineering techniques, are also reviewed. By establishing a thorough understanding of the current state-of-the-art in this domain, the research contributions are presented in the subsequent chapters.

Chapter 3

Malware Dataset Generation and Evaluation

This chapter explores the generation process of malware datasets and their evaluation, aiming to assess the efficacy of malware defense systems.

3.1 Introduction

The Internet is a complex and global networked system that interconnects various networks of networks. The Internet has developed rapidly in the past twenty-five years and is now regarded as a vital organ undergirding the economy of the world. The whole world now can be conceptualized as a 'global village' where one can connect and communicate with any other remote person. The current growth of the Internet is being fueled by the explosion of new sophisticated Internet-enabled devices, rapid growth in the number of ISPs and increasing affordability of Internet services across the globe. Today, a wide range of devices connected to the internet such as cell phones, tablets, phablets, laptops, and netbooks provide hassle-free connectivity. The growth of technology and its services has changed the lifestyles of people. Now, people make use of services available on top of the Internet for numerous purposes. Unfortunately, these services can be exploited by enemies and criminals to fulfill nefarious. They can launch various types of attacks to sweep into interconnected systems by exploiting the vulnerabilities found in different devices and software applications. As a result, they can disrupt ongoing services, steal sensitive information, corrupt important files, and exhaust the resources of servers. To counter such attacks, various defense mechanisms have been proposed in recent

times. It is also important to note here that over the years, various patterns of malware have been introduced from time to time. So, a detection system is required to adapt itself to detect even the most recent malware attacks. The performance of these detection methods is usually established using raw or feature datasets. The non-availability of adequate datasets often becomes a bottleneck in malware research.

3.1.1 Importance of Malware Datasets and Their Desired Characteristics

A number of malware defense systems [21][22][23][24] have been developed to counter malware attacks, each with its own merits and demerits. One common trait of any intrusion detection system is that the system needs to learn a profile or create a model of intrusive traffic or behaviour. This process needs a suitable dataset containing ample examples of intrusive behaviour. However, the nature and type of the dataset that is used for training may vary from one defense system to another. A dataset is represented in a tabular form or a data matrix, where each row represents the samples or observations, and each column contains the value of an attribute or feature for each sample or observation. The caliber of such training datasets plays an important role in the level of performance a detection system can achieve. In the case of malware attacks, a few datasets are available publicly, but most are outdated. As a result, malware detection approaches are difficult to train on the current distribution of malware and hence are likely to not perform accurately and consistently when deployed in real environments. Since the malware patterns change day by day, it is necessary to update a training dataset in order to incorporate the most recent attack experiences. Below we discuss some significant characteristics that a standard dataset should have.

a. *Labels in the data:* The instances in the dataset are labeled into two groups namely malware and benign. The labeling is done manually and requires analysis of raw data. However, it is very difficult to collect substantial numbers of examples of all varieties of malware or benign behaviour compared to raw unlabeled data, and even labeled data can be full of bias. In data labeling, human experts need to pay a high level of attention and effort because any mistake affects the quality of the malware dataset, and ultimately the performance of the trained machine learning model degrades. It is worth mentioning here that the malware data are more difficult to collect than the

benign instances due to various reasons like security, liability, and precautions. Depending on the total presence or absentee or partial presence of label on the dataset, a defense system can built in one of the three modes: supervised, unsupervised, and semi-supervised.

b. *Adequate number of instances:* The dataset should have an adequate number of quality instances in order to ensure the high predictive ability of the trained defense model on unseen data. The number of instances required to improve the performance of the machine learning model depends on the quality of the instances as well as the nature of the undergoing malware.

c. *Adequate number of features:* Features are the derived or observed properties of instances in the dataset. Features represent an instance to a machine learning model so that it can learn a model to make decisions on unseen data. The extracted features on a dataset should be relevant i.e., they should have high correlations with the class labels. A high number of features in a dataset may also create problems. The curse of dimensionality issue may arise in the presence of too many features. As a result, learning models may overfit, and memory requirements and computational costs may also increase. On the other hand, too few features may not be able to distinguish different classes. So, for any particular problem, the dataset should have an adequate number of features such that the class profile is properly defined as well as overfitting can be avoided.

d. *Balance between classes:* The distribution of each class in a dataset should be balanced. That means the number of instances that belong to each class should be almost equal. If the classes are not balanced, the learned model may produce misleading results as it usually tends to favor the bigger classes. The class imbalance problem can be resolved by sampling techniques like oversampling, undersampling or hybrid sampling.

e. *Recency*: The data constituting the dataset should not be outdated. They should be from recent times so that they cover the newer, recently evolved attack strategies. For instance, if the data are outdated and the defense mechanism is trained using such data, it will fail to identify newly evolved unknown attack instances. So, the dataset should be updated from time to time to incorporate recent relevant data.

f. *Lack of inconsistency*: Data in a standard dataset should be consistent. Defense mechanisms trained using inconsistent data are never reliable and will result in wrong decision making.

g. *Relevance*: The features present in the dataset should be relevant to the problem at hand. Irrelevant features are likely to bring down the performance of the defense mechanism. The feature to class relevance should be high, and only then good performance can be expected.

3.1.2 Motivation

To counter malware attacks, the development of malware defense systems is on the rise. However, to measure the efficiency and effectiveness of such defense systems, we need featured malware datasets. At this time, only a very few such featured malware datasets are publicly available. Additionally, with the evolving behavioral patterns of malware, these datasets become outdated, and hence inadequate. Therefore, it is necessary to generate unbiased datasets of malware from time to time to ensure accurate evaluation of malware detection methods.

3.1.3 Contributions

This chapter details the creation of two comprehensive malware feature datasets tailored for two different platforms: Windows and Android. To facilitate this, we developed two specialized frameworks, each consisting of several phases and modules that collaboratively generate the datasets. These datasets are designated as TUMALWD (for Windows) and TUANDROMD (for Android). Furthermore, this research introduces two additional feature datasets. The first is an image-based malware feature dataset, which converts malware binaries into visual formats to leverage image-based analysis techniques. The second is a function call graph (FCG) dataset, which maps the relationships and interactions between various functions within the malware binaries to uncover complex behaviors and patterns. To support the creation and visualization of these datasets, a web-based tool has been developed. This tool allows users to input raw malware binaries and generate the aforementioned datasets. All the datasets developed in this research are publicly available and listed in Table 3.5. This ensures that the broader research community can access and utilize these resources for further studies and advancements in malware detection and analysis.

3.2 Background

Malware plays an important role in all kinds of network intrusions and security attacks. Any software that disrupts user data, computers or networks can be considered malware, including viruses, trojans, worms, rootkits, scareware, and spyware. The concept of a malicious program is not new. The theoretical idea behind malware was introduced by Jon von Neumann in his article *"Theory of self-reproducing automata"* [25]. Till the early 80s, malware was created with benign intent. Most malware was written just for fun or to try out experiments or just to annoy others. Starting from the late 80s, malware have been created with malicious intent, and since then incidents of malware attacks have grown tremendously, and in the present era cyber threats have evolved to a feverish level, hazardously infecting systems and platforms which were not known to be vulnerable earlier. With each passing day, the structures and methods of cyber-attacks are becoming more complex, working with increasing stealth and frequencies of attack. These include clickless threats, and Internet-of-Things (IoT) attacks. For example, the Mirai[1] malware was created to infect Internet-of-Things (IoT) devices like thermostats, webcams, home security systems and routers.

3.2.1 Representation of Malware and Their behavior

To effectively understand and combat malware, it is crucial to recognize the various types of data and files it exploits for attacks. Malware often uses file attachments in emails as common threat vectors, but the danger is not limited to executable files alone. Various file types can harbor threats, including executable files such as .exe, .bat, and .com, which can directly run malicious code and cause immediate harm. Office documents with extensions like .doc, .docx, .xls, .xlsx, .ppt, and .pptx can contain embedded macros or scripts that activate malicious activities when opened. Script files with extensions like .js, .vbs, and .ps1 can execute scripts that perform harmful actions on a system. Compressed files, such as .zip and .rar, can conceal malicious files that execute when decompressed, making them a sneaky vector for malware delivery. Additionally, PDF documents can include embedded scripts or links to malicious websites, posing risks even in seemingly benign files. By understanding these diverse data types and their potential for exploitation, one can better detect and mitigate the various methods malware uses to infiltrate and compromise systems, thereby enhancing overall defense strategies. To build

[1] https://github.com/jgamblin/Mirai-Source-Code

effective defense systems, malware can be represented in various forms such as raw data, feature data, function call graph data, and greyscale images.

1. Raw Data: Raw malware binaries are unprocessed executable codes used by malware to carry out attacks. These binaries can be found in various forms and are not specific to any operating system or file format. They can be distributed through email attachments, compromised websites, or hidden in legitimate software. Once executed, raw malware binaries can exploit system vulnerabilities, install backdoors, steal sensitive information, or disrupt normal operations. Understanding these binaries is crucial for developing effective security measures, as it helps in recognizing and mitigating threats before they cause significant harm.

2. Feature Level Data Features are characteristics extracted from raw data that are relevant to a specific problem, such as malware detection. These features play a crucial role in determining the complexity, readability, and functionality of the data. In the context of malware, features are categorized into the following types:

    - Host-based Features: These features are derived from the malware or benign programs present in system logs or malware analysis reports. Host-based features enumerate the activities that programs can perform to interact with the system. Examples include:
        - *File Operations:* Creating, modifying, or deleting files.
        - *Registry Changes:* Adding or altering registry entries.
        - *Process Behavior:* Starting, stopping, or manipulating system processes.
        - *Memory Usage:* Accessing or modifying memory regions.
        - *System Calls:* Specific calls made to the operating system's API.
    - Network Traffic Features: These features are extracted from network traffic logs generated by programs under analysis. Network traffic features help distinguish between malware and benign programs by analyzing their network behavior. Examples include:
        - *Connection Patterns:* The number and frequency of connections made to various IP addresses.
        - *Data Transfer:* The volume and type of data being sent or received.
        - *Protocols Used:* Specific protocols (e.g., HTTP, FTP) used for communication.

- *Anomalous Behavior:* Unusual traffic patterns, such as connections to known malicious IP addresses or domains.

Understanding and extracting these features from raw data are essential steps in developing robust malware detection systems. These features enable the identification of malicious activities and help in distinguishing malware from benign software, thereby enhancing the effectiveness of security measures.

3. Image data: Malware can be converted into image representations through a process known as binary visualization. In this technique, the binary code of the malware, which is typically a sequence of ones and zeros, is transformed into a pixel-based image. Each binary digit is mapped to a pixel's grey value or intensity value, creating a visual pattern. This conversion preserves the structural and sequential information of the malware, making it possible to analyze the malware's code and behavior in a more visual and interpretable format.

4. Graph data: Malware can be represented as a control flow graph (CFG), which provides a visual and structural depiction of its execution path and behavior. In this representation, the various code blocks, functions, and instructions within the malware are represented as nodes in the graph. The edges between these nodes illustrate the order in which these elements are executed, creating a directed graph that reveals the flow of control and data within the malware.

3.2.2 Benchmark Datasets

In our research, the benchmarking datasets we use are crucial for validating and improving the effectiveness of our proposed methods. These datasets provide the foundation for evaluating the performance of our models in real-world situations, enabling a thorough assessment of their strength and adaptability. Although the benchmark datasets for the evaluation of malware detection systems are limited, we have collected different categories of malware datasets, namely, graph data, image data, and tabular data from trusted sources. This diversification of data sources allows for a more comprehensive and nuanced understanding of the characteristics of malware.

3.2.2.1 MALNET-Graph

MALNET-Graph proposed by Freit et. al. [26] is a large-scale dataset comprising 1,262,024 function call graphs representing 47 types and 696 families of malware. The dataset is stored in edge list format, occupying 443 GB on disk. Each graph, on average, consists of 15,378 nodes and 35,167 edges.

3.2.2.2 MALNET-Image

MALNET-Image proposed by Freit et. al.[27] is a large-scale dataset comprising 1,262,024 images representing 47 types and 696 families of malware.

3.2.2.3 Malimg

Malimg dataset proposed by Natraj et. al.[28] contains a database of 9,458 samples with 25 different malware families.

3.2.2.4 Ransomware Dataset

The dataset proposed by Sgandurra et al. [29] contains 582 samples of ransomware with 11 variants and 942 samples of benign programs. The dataset has 30,962 attributes which represent all instances of both goodware and ransomware present in the dataset.

3.2.2.5 SWaT Dataset

SWaT [30] represents a scaled down version of a real-world industrial water treatment plant producing 5 gallons per minute of water filtered via membrane based ultrafiltration and reverse osmosis units. The main purpose of the dataset, carried out by the research team (Sridhar Adepu and team) was to design secure and safe CPS (Cyber Physical System). SWaT has six main processes corresponding to the physical and control components of the water treatment facility. In total, 946,722 samples comprising of 51 attributes were collected over 11 days. The dataset consists of two labels- “Attack” and “Normal”, where all the different types of attacks are merged into a single class under label - “Attack”.

3.3 TUMALWD: Windows Malware Dataset Creation Framework

The dataset creation framework describes the process and the computing environment needed to create the dataset, which we call TUMALWD. The preparation framework handles three phases, namely data collection and storage, data analysis, and feature engineering. In Phase 1, data are collected and stored. The Second Phase is responsible for data analysis. In Phase 3, we get the dataset TUMALWD as output. Figure 6.3 shows the dataset generation pipeline. The process begins with malware collection from both honeynet environments and the broader internet. Once the malware samples are amassed, the next stage involves malware analysis. This step is crucial for understanding the characteristics and behaviors of the collected malware. Automated tools and manual inspection are utilized to extract features such as file attributes, behavior patterns, and network interactions. Following malware analysis, the dataset undergoes a careful curation process. Irrelevant or redundant samples are filtered out, ensuring that the final dataset is both manageable and highly informative. With the curated and labeled dataset in hand, the evaluation phase begins. Machine learning models are trained on a subset of the dataset and then evaluated on a separate, unseen portion.

The entire pipeline, from malware collection and analysis to dataset curation and machine learning model evaluation, is iterative. As the threat landscape evolves, the dataset creation pipeline is revisited to incorporate new samples and adapt to emerging trends, ensuring that the machine learning models remain effective in addressing the latest malicious samples.

3.3.1 Phase 1: Data Collection and Storage

For collecting malware, an automated honeynet system is set up to automatically store the collected malware binaries in a centralized server. The whole system is based on a client-server architecture where the client module is responsible for data collection and the server module is used for data storage. A generic architecture of the data collection framework is shown in Figure 3-2.

a). *Honeynet Client*: On the client side, two virtual honeypots are set up and configured in a workstation with Linux-based OS using a Virtual box. This virtualization offers an emulated environment and also mimics the operating

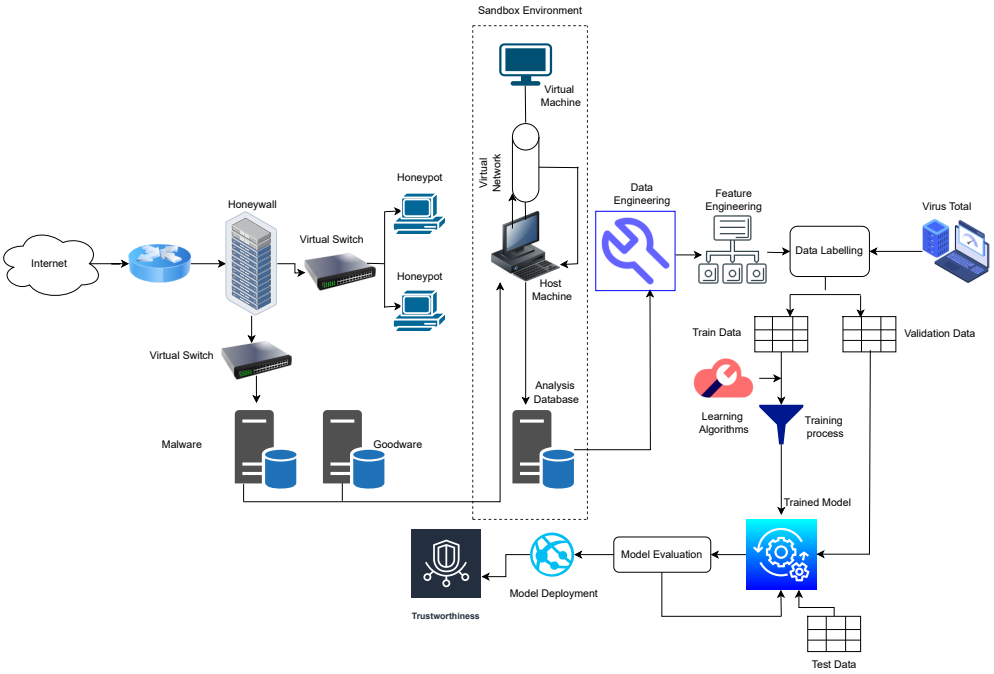

Figure 3-1: Dataset Generation and Evaluation Pipeline

systems services and ports. The honeypots used in the system are run on the Windows platform. As shown in the figure, a honeywall is used as a gateway for the honeypots. This allows for complete control of all the inbound and outbound traffic. During the operation period of five months i.e., August 2019 to December 2019, 4000 malware binaries were collected.

b). *Honeynet Server*: This module stores all the malware binaries and network traffic files collected at the honeynet clients. Further, all the collected binaries are transferred to an Analysis Server for further processing and analysis.

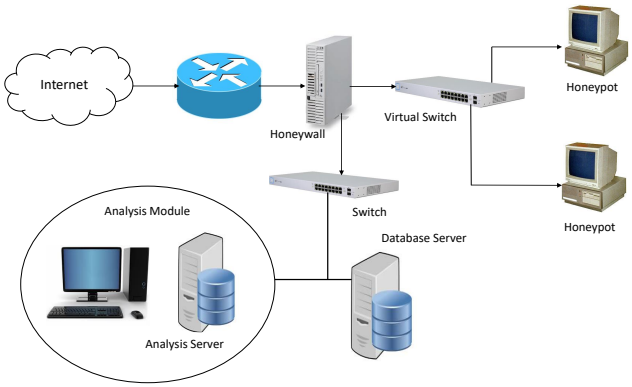

Figure 3-2: Testbed Architecture for malware collection

3.3.2 Phase 2: Data Analysis

After the Data collection and Storage phase, the collected malware binaries are transfered to the Data Analysis phase where thorough malware analysis is conducted. To accomplish the tasks of Phase 2, two modules come into action, namely Analysis Client and Analysis Server respectively responsible for malware analysis and storage of all analysis reports in a database server for further processing and feature extraction. Steps followed to accomplish the tasks in Phase 2 are shown in figure 3-3.

a). *Analysis Client* : This module receives the input from the Honeynet Server module in the form of binary programs. For the analysis of the collected malware binaries, we set up and configured an open source malware analysis tool called the Cuckoo Sandbox [2]. The Cuckoo Sandbox is an automated system which can analyze different types of files in a realistic but isolated environment. Initially, we installed the Cuckoo Sandbox in a worksation where Ubuntu 16 is used as the host operating system. For the creation of an isolated environment, we used Virtual Box where Winodws7 is installed as a guest operating system to carry out all analysis tasks. For the analysis, the Cuckoo Sandbox is fed with malware binaries and in return, it genertaes a complete behavioral report for each malware binary. In addition to Cuckoo Sandbox, we also installed an API monitor [3] in the guest operating system to monitor and log the API calls made by malware applications.

b). *Analysis Server* : This component receives all the analysis reports as input from the *Analysis Client* component and stores them in a database server for further processing and feature extraction. It also receives all log files of the API monitor as input and stores them in the database server.

3.3.3 Phase 3: Feature Engineering

Phase 3 is the final and main part of our dataset generation framework. To accomplish the tasks of Phase 3, two components come into action, namely preprocessing and feature extraction.

[2]https://www.cuckoosandbox.org/
[3]https://www.apimonitor.com/

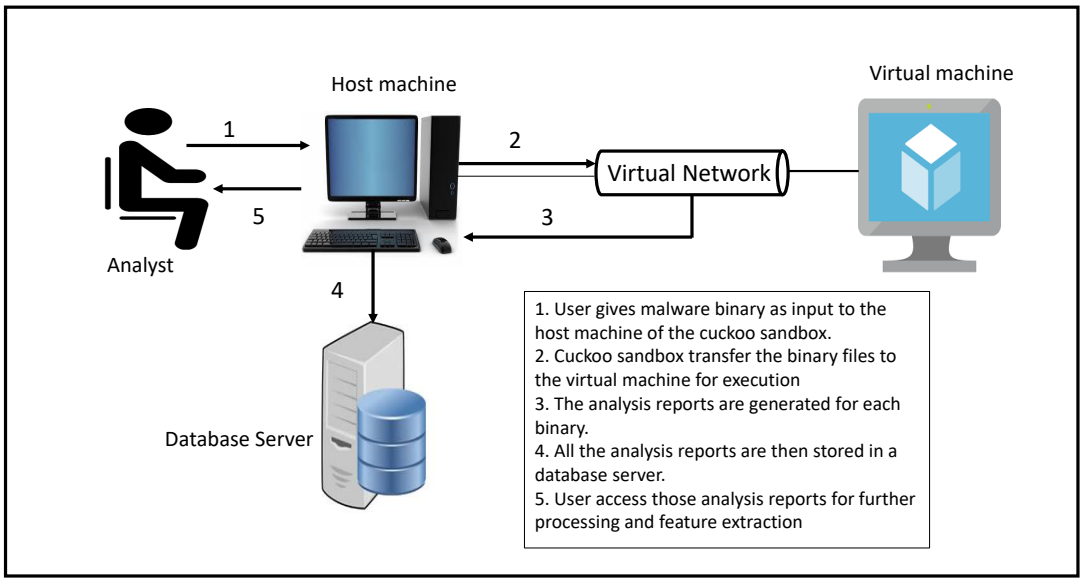

Figure 3-3: Steps in data analysis phase

a). *Preprocessing*: In the preprocessing component, we discard all the analysis report whose content is empty, i.e., those malware binaries that failed to execute during analysis. For our conveninece and to carry forward the tasks of preprocessing, we extract meaningful contents from the analysis reports and stores them in a separate file. To accomplish this task, we wrote C programs to extract information from the API calls for which the status value is success. In other words, our programs takes the text file of the analysis report as an input and searches for the status value of each API block. Once it encounters the status tag and if its value is "SUCCESS", it writes all the API call information to an another output file and this process is goes on until the end of the file. Figure 3-4 is an example of API call information of a binary file obtained during analysis. As seen in the figure, the binary file successfully created a file in the system during execution.

{'api': 'CreateFileW', 'arguments':
[{'name': 'lpFileName', 'value': 'C:\\WINDOWS\\system32\\svchost.exe'}, {'name': 'dwDesiredAccess', 'value':
'GENERIC_READ'}], 'repeated': 1, 'return': '0x000000b4', 'status': 'SUCCESS', 'timestamp': '20111219100546.734'},
{'api': 'CreateProcessA', 'arguments':
[{'name': 'lpApplicationName', 'value': '(null)'},
{'name': 'lpCommandLine', 'value': 'svchost.exe'}], 'repeated': 0, 'return': '1548', 'status': 'SUCCESS', 'timestamp':
'20111219100546.734'},
{'api': 'VirtualAllocEx', 'arguments':
[{'name': 'th32ProcessID', 'value': '1548'},
{'name': 'szExeFile', 'value': 'svchost.exe'}, {'name': 'lpAddress', 'value': '0x00000000'}, {'name': 'dwSize', 'value': '0'},
{'name': 'flAllocationType', 'value': '0x00003000'}, {'name': 'flProtect', 'value': '0x00000040'}], 'repeated': 0, 'return': '',
'status': 'FAILURE', 'timestamp': '20111219100546.734'},
{'api': 'ExitProcess', 'arguments':
[{'name': 'uExitCode', 'value': '0x00000000'}], 'repeated': 0, 'return': '', 'status': '', 'timestamp': '20111219100546.744'}],
'first_seen': '20111219100536.679', 'process_id': '764', 'process_name': 'binary.exe'}]

Figure 3-4: Example of API call information

b). *Feature Extraction* : In the Feature Extraction phase, two categories of fea-

tures, namely, host based and network based features, are extracted from the preprocessed analyis reports. Host based features contain three feature sets where the first set contains the API call sequences, the senond set comprises the successful API call status and the third set contains the frequency of each API call. On the other hand, network based feature set contain all the flow information of malware binary applications.

3.3.4 TUMALWD: The Dataset and Its Characteristics

The processes and phases involved in the dataset creation framework have been discussed in detail in the previous sections. The output of the framework is the dataset TUMALWD comprising of two feature sets, namely host based and network based features. Host based and network based features are for the system based and the network based indicators, respectively.

3.3.4.1 Host-based features

For host based features, we consider Windows API functionality that helps distinguish between malware and benign programs. We extract all critical API functions that malware used successfully in order to interact with the operating system's libraries. These features are extracted from the API call information. The list of top ranked feature categories are enlisted in Table 3.1. The ranks are calculated using three feature selectors [31][32] [33].

Table 3.1: List of top ranked feature categories for TUMALWD

Feature Rank	Feature Category
1	*RegistryKeysOperations*
2	*ProcessCalls*
3	*Strings*
4	*Filesoperations*
5	*Fileextensions*
6	*Directoryoperations*
7	*Droppedfiles*

3.3.4.2 Network traffic features

To extract network level features, a malware program needs to perform activities over the network during execution. The network activities of malware are captured

by executing them in the Cuckoo sandbox. Additionally, we collect network traces that are captured in the honeynet. The traces are analyzed using an open source tool called Wireshark. The raw traces are then preprocessed and filtered to extract different features. We extract 27 different flow-based features from the traces. To extract these features we wrtie Python scripts and also use an open source tool called Flowtbag [4]. These features can be specifically classified into basic features, time-based and byte-based features.

1. Basic features: Unlike the standalone Malware, some variations perform malicous activities based on the command they receive from a master program. Consequently, the infected system needs to establish a connection to the master machine. Basic network features help identify if the compromised victim system performs any malicious activity over the network. For example, if the victim system tries to establish an unwanted connection with a blacklisted website, it can easily be detected based on the destination IP of the blacklisted site.

2. Time-based features: Time-based features help distinguish malicious network connections from legitimate ones. For example, the duration of the flow, i.e., the start and end time of the flow can play a useful role in identifying malware compromised hosts as most follow similar communication patterns with their masters, and the respective durations are almost similar.

3. Byte-based features: By monitoring the amount of data exchange between two hosts, it is possible to infer whether the communication link is malicious or not.

3.3.4.3 Characteristics of TUMALWD

As discussed in Section 3, the following are the particular characteristics of the TUMALWD dataset that we create.

1. *Labels in the data:* The TUMALWD dataset has two labels, namely benign and malware. The benign instances make up the benign class, while the malicious instances make up the malware class.

2. *Number of instances:* There are a total of 5400 instances for the host-based features, of which 4000 instances belong to the malware class and the rest

[4]https://github.com/DanielArndt/flowtbag

Table 3.2: Network level features

Category	Feature name	Feature description
Basic features	Src IP	Source IP
	Src port	Source port
	Dst IP	Destination IP
	Dst port	Destination port no
	Total_pkts	Total packets in the forward direction
	Proto	The protocol
Byte based features	total_fvolume	Total bytes in the forward direction
	total_bvolume	Total bytes in the backward direction
	min_fpktl	Smallest packet size (forward direction)
	mean_fpktl	Average size of the packets (forward direction)
	max_fpktl	Maximum size of the packets (forward direction)
	min_bpktl	Least packet size (backward direction)
	mean_bpktl	Average size of the packets (backward direction)
	max_bpktl	Maximum size of the packets (backward direction)
Time based features	Duration	The flow duration
	min_fiat	The least amount of times b/w two packets (forward direction).
	mean_fiat	The average amount of times b/w two packets (forward direction).
	max_fiat	The highest amount of times b/w two packets (forward direction).
	std_fiat	The standard deviation from the mean amount of times b/w two packets sent in the forward direction
	min_biat	The least amount of times b/w two packets (backward direction)
	mean biat	The average amount of times b/w two packets (backward direction)
	max_biat	The max amount of times b/w two packets sent in the backward direction
	std_biat	The standard deviation from the average amount of times b/w two packets (backward direction)
	min_active	The least amount of time of the active flow
	mean_active	The average amount of time of the active flow
	max_active	The max amount of time of the active flow
	std_active	The standard deviation of the active flow

1400 instances belong to the benign class. For network-based features, there are a total of 3000 instances, of which, 1600 instances are malware and the rest 1400 instances are benign.

3. *Number of features:* For host-based features, all the successful API calls are extracted as features. A total of 7545 calls are extracted. Additionally, 27 network-based features are extracted.

4. *Balance between the classes:* The dataset is not perfectly balanced i.e., the dataset does not have an equal number of instances for both classes. This slight imbalance can be easily handled by collecting more instances or sampling techniques. The proposed dataset creation framework can handle such adaptations.

5. *Recency:* The data collected for the creation of TUMALWD are recent (dated to late 2019). As and when new malicious instances are available, the dataset can be updated accordingly.

6. *Relevance:* The extracted features help distinguish between malware and

benign programs. To the best of our knowledge, these features are relevant differentiating between the two.

3.4 TUANDROMD: The Proposed Android Malware Dataset Creation Framework

The dataset creation framework describes the whole preparation process for the created dataset: TUANDROMD. The preparation framework is divided into three phases, namely data collection, data analysis, and feature engineering. In Phase 1, benign and malware Android applications are collected for further analysis. In the second phase, data analysis is carried out on the data collected in Phase 1. Finally, Phase 3 takes the output of Phase 2 as input and performs feature engineering to output the final dataset, which we call TUANDROMD.

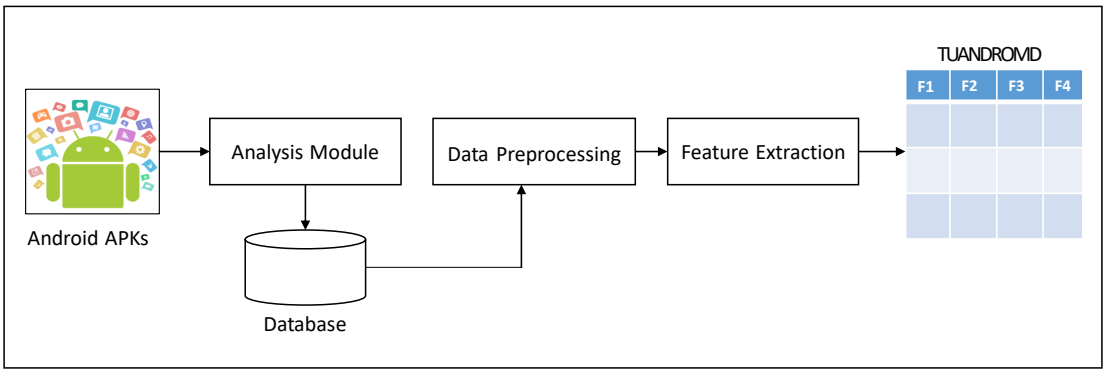

Figure 3-5: TUANDROMD: Dataset creation framework

3.4.1 Phase 1: Data Collection and Storage

For developing a featured dataset of Android malware, we need raw malware and benign applications. For this, we collected 24,553 raw malware binaries from [34]. These malware binaries are further categorized into 135 varieties among 71 malware families. Additionally, we gathered the top 1000 Android applications from Google Play as benign applications. Finally, all the collected Android applications are stored in a database server for further processing and analysis.

3.4.2 Phase 2: Data Analysis

Unlike PC-based malware analysis, we have performed manual analysis on the collected Android malware using a set of tools–Androguard[5], ApkInspector[6], ApkAnalyser[7] and Smali-CFGs[8]. For analysis, we consider each variant of malware in each family. All analysis tasks are carried out in an isolated environment. For the creation of isolated environment, we used Virtual Box where Winodws7 is installed as a guest operating system in which all the analysis tasks are carried out.

3.4.3 Phase 3: Feature Engineering

In the feature engineering phase, two categories of features namely permission-based and API-based features are extracted from the analysis results. For permission-based features, we extract all the install time and runtime permissions required for an Android application to run smoothly on the device. For API-based features, we extract all the APIs used by an application to carry out all the tasks for the users.

3.4.4 TUANDROMD: The proposed dataset and its characteristics

The processes and phases involved in the dataset creation framework has been discussed in detail in the previous sections. The output of the framework is the dataset TUANDROMD comprising of two feature sets namely permission-based and API-based features.

1. Permission-based features:

   Android has a built-in defense system which is termed as permission-based system. This system defines a set of actions that an app is allowed or not allowed to perform. That permission that requires user confirmation can be exploited by the attackers to harm or compromise the system. These permission-based features help to distinguish between malicious and benign

[5]https://github.com/androguard/androguard
[6]https://github.com/honeynet/apkinspector/
[7]https://github.com/sonyxperiadev/ApkAnalyser
[8]https://github.com/EugenioDelfa/Smali-CFGs

apps. These features are extracted from our analysis report using several C routines. A total of 178 features are extracted out of which 15 top most features are enlisted in table 3.3.

Table 3.3: List of top-ranked features for TUANDROMD

Feature Rank	Feature Name
1	$SEND\_SMS$
2	$RECEIVE\_BOOT\_COMPLETED$
3	GET_TASKS
4	$Ljava/net/URL; -> openConnection$
5	$VIBRATE$
6	$WAKE\_LOCK$
7	$KILL\_BACKGROUND\_PROCESSES$
8	$SYSTEM\_ALERT\_WINDOW$
9	$ACCESS\_WIFI\_STATE$
10	$DISABLE\_KEYGUARD$
11	$Landroid/location/LocationManager;$ $-> getLastKnownLocation$
12	$READ\_PHONE\_STATE$
13	$RECEIVE\_SMS$
14	$CHANGE\_WIFI\_STATE$
15	$WRITE\_EXTERNAL\_STORAGE$

2. API-based features:

   The Android platform provides a framework API that consists of a core set of packages and classes. Since most of the applications are dependent on a large number of API calls, it seems all the more practical and sensible to use API calls of each application as a feature to characterize and differentiate malware from benign applications. These features are also extracted from the analysis reports using several C routines. A total of 241 unique API calls are extracted.

3.4.4.1 Characteristics of TUANDROMD

As discussed in Section 3, the following are the particular characteristics of the TUANDROMD dataset that we created.

1. *Labels in the data:* The TUANDROMD dataset has 72 labels where 71 labels represent the whole malware family and the remaining one label belongs to the normal class.

2. *Number of instances:* There are a total of 25,553 instances for both the permission and API-based features, of which 24,553 instances belong to the malware class and the rest 1000 instances belong to the benign class.

3. *Number of features:* For permission-based features, all the permissions used by the applications are extracted as features. A total of 178 features are extracted. Similarly, for API-based features, a total of 241 features are extracted.

4. *Balance between the classes:* The dataset is not perfectly balanced i.e., the dataset does not have an equal number of instances for both classes. This slight imbalance can be easily handled by collecting more instances or sampling techniques. The proposed dataset creation framework can handle such adaptations.

5. *Recency:* The data collected for the creation of TUANDROMD are recent. As and when new malicious and normal Android applications are available, the dataset can be updated accordingly.

6. *Relevance:* The extracted features help distinguish between malware and benign Android applications. To the best of our knowledge, these features are relevant differentiating between the two.

3.5 Performance Evaluation and Validation

For evaluating the performance and also to form a benchmark on these datasets, we used a total of five classifiers. For each classifier, the K-Fold cross-validation is used where the value of K=10. The classification results are enlisted in table 3.4. The reason we use classification algorithms for evaluation to assess the effectiveness of the datasets so that machine learning-based malware defense systems can be developed using these datasets.

Table 3.4: Benchmark on TUMALWD and TUANDROMD

Classifier	Test Accuracy(%)	
	TUMALWD	TUANDROMD
Random Forest	98.4	98.7
Extra Tree	97.6	98.8
Ada Boost	98.5	97.9
Xg Boost	97.8	97.8
Gradient boosting	96.3	97.4

3.6 Additional Malware Feature Datasets

In addition to the Windows and Android malware datasets, two more feature datasets have been introduced to this research. These include an image-based malware feature dataset and a function call graph (FCG) dataset, both derived from collected raw malware binaries.

3.6.1 Image-Based Malware Feature Dataset

The image-based dataset converts malware binaries into grayscale images, capturing their structural patterns for analysis using image processing and deep learning. Key features of this dataset include:

- **Image Conversion**: Each byte of the malware binary is represented as a pixel, with values from 0 to 255.
- **Dataset Size**: It contains 20,000 samples from various malware families, providing a diverse set for analysis.
- **Feature Extraction**: Convolutional neural networks (CNNs) extract features from these images, identifying intricate patterns within the malware binaries.

Details on this dataset, including image conversion methods, CNN architectures, and model performance, are covered in Chapter 8.

3.6.2 Function Call Graph (FCG) Dataset

The FCG dataset extracts function calls from raw malware binaries and represents them as graphs, capturing their execution flow for analysis. Key features of this dataset include:

- **Nodes and Edges**: Nodes are functions, and edges represent the calls between them.
- **Dataset Size**: It contains 20,000 samples from various malware families, providing a diverse set for analysis.