San Jose State University

SJSU ScholarWorks

Master's Projects Master's Theses and Graduate Research

Fall 2022

Adversarial Attacks on Android Malware Detection and Classification

Srilekha Nune
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Information Security Commons

Recommended Citation
Nune, Srilekha, "Adversarial Attacks on Android Malware Detection and Classification" (2022). *Master's Projects*. 1196.
DOI: https://doi.org/10.31979/etd.gjhb-v87s
https://scholarworks.sjsu.edu/etd_projects/1196

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Adversarial Attacks on Android Malware Detection and Classification

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Srilekha Nune

December 2022

© 2022

Srilekha Nune

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

Adversarial Attacks on Android Malware Detection and Classification

by

Srilekha Nune

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

December 2022

Dr. Mark Stamp	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science
Dr. William Andreopoulos	Department of Computer Science

ABSTRACT

Adversarial Attacks on Android Malware Detection and Classification

by Srilekha Nune

Recent years have seen an increase in sales of intelligent gadgets, particularly those using the Android operating system. This popularity has not gone unnoticed by malware writers. Consequently, many research efforts have been made to develop learning models that can detect Android malware. As a countermeasure, malware writers can consider adversarial attacks that disrupt the training or usage of such learning models. In this paper, we train a wide variety of machine learning models using the KronoDroid Android malware dataset, and we consider adversarial attacks on these models. Specifically, we carefully measure the decline in performance when the feature sets used for training or testing are contaminated. Our experimental results demonstrated that elementary adversarial attacks pose a significant threat in the Android malware domain.

ACKNOWLEDGMENTS

I thank Dr. Mark Stamp for his ongoing advice and assistance throughout my graduate career. He offered me some insightful advice whenever I got stuck while working on this project. Working on his class projects and taking his machine learning courses were enjoyable. His extensive experience was beneficial. In addition, his books on information security and machine learning made me a huge fan.

I want to thank Dr. Thomas Austin, a committee member, for agreeing to join in the committee after I requested. I first learned about him when I enrolled in his Advanced Programming Languages course. Wow, what a well-organized course with brief lectures on many different languages. He is highly kind and readily available to students via email and canvas messages. I had a great time in his course.

Thank you to Dr. William Andreopoulos, another committee member, for agreeing to join my committee at the last moment. I would want to thank him for his time and insightful feedback. He has a fantastic personality and is realistic.

I would want to end by thanking my complete family for their assistance during my master's. Without their motivation and help, I would not have come this far. A special thanks go out to my kids for being understanding and not bugging me too much throughout this adventure. Many thanks to my SJSU instructors for their guidance and help throughout the course.

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

2 Background 5

2.1 Related Work 6

2.1.1 Android Malware Detection and Classification 6

2.1.2 Adversarial Attacks on Android Malware 8

3 Methodology 10

3.1 Adversarial Attacks 10

3.1.1 Data Poisoning Attack 10

3.1.2 Evasive Attack 11

3.2 Techniques for Classification 11

3.2.1 Multi-Layer Perceptron (MLP) 11

3.2.2 Logistic Regression (LR) 12

3.2.3 Support Vector Machine (SVM) 13

3.3 Metrics to Measure Performance 14

4 Implementation 16

4.1 Dataset 16

4.1.1 Data Preprocessing 17

4.2 Implementation of Adversarial Attacks 20

4.2.1 Label Flip on Binary Classification 20

4.2.2 Label Flip on Multi-Class Classification 22

4.2.3 Evasive Attack on Binary Classification 22
4.3 Implementation of Techniques 22
5 Results 23
5.1 Label Flip Attack on Binary Classification 23
5.1.1 Support Vector Machine 23
5.1.2 Logistic Regression 24
5.1.3 Multi-Layer Perceptron 25
5.2 Label Flip Attack on Multi-Class Classification 27
5.2.1 Support Vector Machine 27
5.2.2 Logistic Regression 30
5.2.3 Multi-Layer Perceptron 32
5.3 Evasive Attack on Binary Classification 35
5.3.1 Support Vector Machine 35
5.3.2 Logistic Regression 38
5.3.3 Multi-Layer Perceptron 39
6 Conclusions and Future Work 42
LIST OF REFERENCES 44
APPENDIX
A Label Flip Attack on Binary Classification 47
B Label Flip Attack on Multi-Class Classification 54
C Evasive Attack on Binary Classification 61

LIST OF TABLES

1 Families Count in KronoDroid 20

2 Selected Features to Perform Evasive Attack 21

3 Hyperparameters for Label Flip Attack on Binary Classification . 23

4 Comparison of Accuracy with Label Flip Attack for Binary 27

5 Hyperparameters for Label Flip Attack on Multi-Class Classification 28

6 Recall, Precision and F1-score before Attack with SVM 29

7 Recall, Precision and F1-score after 10% Attack with SVM 30

8 Recall, Precision and F1-score before Attack with LR 31

9 Recall, Precision and F1-score after 10% Attack with LR 32

10 Recall, Precision and F1-score before Attack with MLP 33

11 Recall, Precision and F1-score after 10% Attack with MLP 33

12 Comparison of Accuracy with Label Flip Attack for Multi-Class . 35

13 17 Important Features to Make Malware as Benign 37

14 Hyperparameters for Evasion Attack on Binary Classification . . 38

15 Comparison of Accuracy with Evasive Attack 41

LIST OF FIGURES

1	Architecture for Poisoning Attack	3
2	MLP with Two Hidden Layers	12
3	SVM with Hyperplane	14
4	Process of Dataset Creation	16
5	KronoDroid Samples Count	17
6	KronoDroid Top Families	18
7	Accuracy of Binary Models without any Attack	24
8	SVM Binary Classification with Label Flip Attack	25
9	LR Binary Classification with Label Flip Attack	26
10	MLP Binary Classification with Label Flip Attack	27
11	Label Flip Attack on the Models	28
12	Accuracy of Multi-Class Models without any Attack	29
13	SVM Multi-Class Classification with Label Flip Attack	31
14	LR Multi-Class Classification with Label Flip Attack	32
15	MLP Multi-Class Classification with Label Flip Attack	34
16	Label Flip Attack on the Multi-Class Models	34
17	Feature Importance for Selected 28 Features	36
18	Accuracy of Models before Evasive Attack	37
19	SVM with Evasive Attack	38
20	LR with Evasive Attack	39
21	MLP with Evasive Attack	40

22 Evasive Attack on the Models 41
A.23 SVM Confusion Matrix before Label Flip Attack 47
A.24 SVM Confusion Matrix with 10% Label Flip Attack 47
A.25 SVM Confusion Matrix with 20% Label Flip Attack 47
A.26 SVM Confusion Matrix with 30% Label Flip Attack 47
A.27 SVM Confusion Matrix with 40% Label Flip Attack 48
A.28 SVM Confusion Matrix with 50% Label Flip Attack 48
A.29 SVM Confusion Matrix with 60% Label Flip Attack 48
A.30 SVM Confusion Matrix with 70% Label Flip Attack 48
A.31 LR Confusion Matrix before Label Flip Attack 49
A.32 LR Confusion Matrix with 10% Label Flip Attack 49
A.33 LR Confusion Matrix with 20% Label Flip Attack 49
A.34 LR Confusion Matrix with 30% Label Flip Attack 49
A.35 LR Confusion Matrix with 40% Label Flip Attack 50
A.36 LR Confusion Matrix with 50% Label Flip Attack 50
A.37 LR Confusion Matrix with 60% Label Flip Attack 50
A.38 LR Confusion Matrix with 70% Label Flip Attack 50
A.39 LR Confusion Matrix with 80% Label Flip Attack 51
A.40 MLP Confusion Matrix before Label Flip Attack 51
A.41 MLP Confusion Matrix with 10% Label Flip Attack 51
A.42 MLP Confusion Matrix with 20% Label Flip Attack 51
A.43 MLP Confusion Matrix with 30% Label Flip Attack 52
A.44 MLP Confusion Matrix with 40% Label Flip Attack 52

A.45 MLP Confusion Matrix with 50% Label Flip Attack 52

A.46 MLP Confusion Matrix with 60% Label Flip Attack 52

A.47 MLP Confusion Matrix with 70% Label Flip Attack 53

A.48 MLP Confusion Matrix with 80% Label Flip Attack 53

B.49 SVM Confusion Matrix before Label Flip Attack 54

B.50 SVM Confusion Matrix with 10% Label Flip Attack 54

B.51 SVM Confusion Matrix with 20% Label Flip Attack 54

B.52 SVM Confusion Matrix with 30% Label Flip Attack 54

B.53 SVM Confusion Matrix with 40% Label Flip Attack 55

B.54 SVM Confusion Matrix with 50% Label Flip Attack 55

B.55 SVM Confusion Matrix with 60% Label Flip Attack 55

B.56 SVM Confusion Matrix with 70% Label Flip Attack 55

B.57 LR Confusion Matrix before Label Flip Attack 56

B.58 LR Confusion Matrix with 10% Label Flip Attack 56

B.59 LR Confusion Matrix with 20% Label Flip Attack 56

B.60 LR Confusion Matrix with 30% Label Flip Attack 56

B.61 LR Confusion Matrix with 40% Label Flip Attack 57

B.62 LR Confusion Matrix with 50% Label Flip Attack 57

B.63 LR Confusion Matrix with 60% Label Flip Attack 57

B.64 LR Confusion Matrix with 70% Label Flip Attack 57

B.65 LR Confusion Matrix with 80% Label Flip Attack 58

B.66 MLP Confusion Matrix before Label Flip Attack 58

B.67 MLP Confusion Matrix with 10% Label Flip Attack 58

B.68 MLP Confusion Matrix with 20% Label Flip Attack 58
B.69 MLP Confusion Matrix with 30% Label Flip Attack 59
B.70 MLP Confusion Matrix with 40% Label Flip Attack 59
B.71 MLP Confusion Matrix with 50% Label Flip Attack 59
B.72 MLP Confusion Matrix with 60% Label Flip Attack 59
B.73 MLP Confusion Matrix with 70% Label Flip Attack 60
B.74 MLP Confusion Matrix with 80% Label Flip Attack 60
C.75 SVM Confusion Matrix before Evasive Attack 61
C.76 SVM Confusion Matrix with 10% Evasive Attack 61
C.78 SVM Confusion Matrix with 30% Evasive Attack 61
C.80 SVM Confusion Matrix with 50% Evasive Attack 62
C.81 SVM Confusion Matrix with 60% Evasive Attack 62
C.82 LR Confusion Matrix before Evasive Attack 62
C.83 LR Confusion Matrix with 10% Evasive Attack 63
C.84 LR Confusion Matrix with 20% Evasive Attack 63
C.85 LR Confusion Matrix with 30% Evasive Attack 63
C.86 LR Confusion Matrix with 40% Evasive Attack 63
C.87 LR Confusion Matrix with 50% Evasive Attack 64
C.88 LR Confusion Matrix with 60% Evasive Attack 64
C.89 MLP Confusion Matrix before Evasive Attack 64
C.90 MLP Confusion Matrix with 10% Evasive Attack 64
C.91 MLP Confusion Matrix with 20% Evasive Attack 65
C.92 MLP Confusion Matrix with 30% Evasive Attack 65

C.93 MLP Confusion Matrix with 40% Evasive Attack 65

C.94 MLP Confusion Matrix with 50% Evasive Attack 65

CHAPTER 1

Introduction

Smartphone usage has been increasing day by day. Most customers are replacing their mobile phones with smartphones. Mobile phones are communication and computing devices [1]. The most popular platform for smartphones is Android. Other than the United States, in the remaining countries, the sales market for Android is high. Especially the Android OS is playing high sales in the current market. Smartphones become access points for almost everything, like monthly payments, online shopping, controlling smart cars, smart doors at the house, and so many others. So, smartphones are the single device with all the treasures.

The fast growth of Android usage worldwide and high market share made attackers focus on it. Malicious attackers started attacking smartphones through malicious applications. Google's Android market is the online market that provides software for Android smartphones [2]. It has both official and unofficial repositories. The Android market allows developers to upload their applications without using any certificate authority. So this makes it easy for the attackers to upload the application with malware and spread it to unregistered repositories.

Many applications are modified, inserted malicious parts in them, and distributed in Google's Android Market, which makes spread easy. The Android platform has a robust permission system [3]. It will restrict the installation of malicious applications by warning or asking for permission to install. Some activities, like sending Short Message/Messaging Service (SMS) from a malicious application, also require consent from the user. But the users will ignore the warning and give the permissions which allow the malicious application to achieve its goal. The malicious applications use advanced techniques. There are many variations in malicious applications.

- Some applications will download malicious code after some time of installation.

- Hide the malicious code inside the applications using obfuscation.
- Use the same name, icon, and version of the original application with the malware functions in it.

McAfee Mobile Treat Report issued in 2020 says that the number of malicious applications is increasing, and most are on the Android operating system [4]. Therefore, mobile targeting malware is rising faster than computer/PC targeting malware. Moreover, day by day, the attackers are improvising the application with malware based on the current trend. Attackers can induce malware into the applications through repackaging, drive-by downloads, and update attacks [1].

- `Repackaging:` Malware developers take and download the original application, dissemble it, patch it with the malware, re-assemble and upload it to the actual market or some other sites. The users will download these applications, install them, and get caught by the malware.
- `Drive-by download:` This method will not upload the malware directly. But it will entice the users to download other applications or features.
- `Update attack:` The same repackaging process is done here but filled with the update component, not the whole malware code. It will download the malware when the application is running. So, it is little harder to detect and cannot be done by static analysis.

Researchers have developed various techniques to combat Android malware. However, at the same time, attackers are introducing adversarial attacks in mobile applications to escape from the detection model. In this study, adversarial attacks [5] are our primary emphasis, and we examine how well the models continue to function even after applying these attacks. Support Vector Machine, Logistic Regression, and Multi-Layer Perceptron are the models that were used in our project. Our primary adversarial strategy in this project involved changing class labels to create adversarial

samples. The system architecture for this adversarial attack is illustrated in Figure 1.

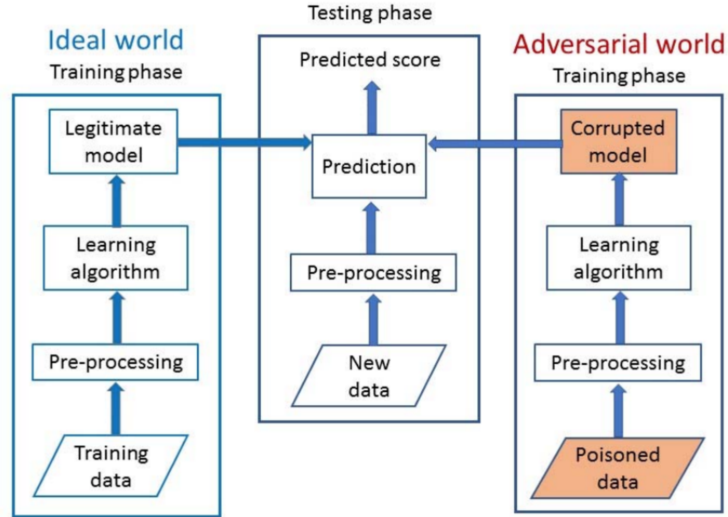

Figure 1: Architecture for Poisoning Attack [6]

The second included altering the components of malware testing samples such that malware detection models would view them favorably. For example, instead of altering the training data as in a previous attack, we will alter the new data we are entering into the model. Our primary attention was on these two categories of strategies.

The features employed are just as crucial to how well the malware detection exercise turns out as the methods and algorithms used to carry out the detection. Features extracted from Android applications can be divided into static and dynamic. Static features are collected directly from the source code and binaries without executing the application. Examples of static features include Application Programming Interface (API) calls, permissions, intents, opcodes, and control flow graphs. Dynamic features are extracted by running or emulating an application. Examples of dynamic features include network activity, system calls, dynamic API calls, register changes, instruction

traces, etc. Extracting dynamic features is generally more time and resource-intensive, but dynamic features are typically more immune to standard obfuscation techniques. So, we use a dataset consisting of dynamic and static features.

This paper is structured as follows. Chapter 2 discusses relevant background topics and related work on adversarial attacks and malware evolution. Chapter 3 presents a brief review of the machine learning algorithms and adversarial attacks we employ. Dataset related details and experiment setup will be discussed in Chapter 4. Finally, results are covered in Chapter 5, while the conclusion and potential future work are discussed in Chapter 6.

CHAPTER 2

Background

This section covers the relevant research that has been done so far on Android malware detection, critical characteristics of Android malware, and adversarial attacks in this area.

There are two approaches for analyzing and detecting Android malware: static analysis and dynamic analysis [7]. Hybrid analysis can be done by using both static and dynamic analysis. Static analysis is done by inspecting source code and binaries for malicious patterns; no need to execute the application. Most static features can be extracted by inspecting the package, accessing the manifest file, and decompiling the source code.

The most common features of static analysis are API calls, Permissions, Intent, Opcodes, Control Flow, etc. But obfuscated malware cannot be detected by the static. The benefit is high code coverage [8]. It is also called signature-based detection. However, the signature-based software must maintain the repository of the known attack's signature repository and keep updating it, which is the main disadvantage of static analysis.

Other is dynamic analysis, also called behavior-based detection. Features are captured by running the applications on the emulator or smartphones in an isolated environment. Some features are network activity, system calls, dynamic API calls, register changes, and instruction traces. However, it has less code coverage.

Nowadays, malicious code is able to find whether the applications are run on the emulator or actual device and used by real users or automated tools. This makes the dynamic analysis weaker. By triggering some activities to the real device or emulator, monitoring agents capture the features like network traffic, cryptographic functions, system calls, and so on.

Compared to dynamic, static is relatively simple, easy to implement, and works for most known attacks. To overcome the disadvantages of static and dynamic analysis, several authors researched hybrid analysis, which combines both static and dynamic analysis. It has the advantages of both analysis and consumes a lot of resources and time.

2.1 Related Work

We discuss pertinent research on Android malware in this section. First, we go into the specifics of Android malware detection and begin working on its feature analysis. Second, we discuss the works on adversarial attacks.

2.1.1 Android Malware Detection and Classification

A lightweight client called "Crowdroid" to monitor the Linux Kernel system calls in Android mobiles is developed [2]. System calls are the most critical features, as malware applications will perform some or other operations on the system. A dedicated server is used to capture data and analyze it [2]. Deployed this framework into Google's Market. As more customers download this onto their smartphones, more data about the system calls will be collected for each application and even warn the user if any abnormal activity happens in the smartphone. This framework distinguishes the applications with the same version and name but behaves differently.

A comprehensive static analysis on Android malware is performed and proposed Drebin - a lightweight method to detect Android malware by gathering as many static features as possible from the application dex code and manifest [3]. The features are extracted by using the Android Asset Packaging Tool. All the features are formulated in the vector space. Applied Support Vector Machines (SVM) to separate the benign and malicious applications from the vectors. It will provide the user with appropriate messages about the application, whether benign or malicious, how it was concluded

as malicious, and which feature makes it malicious.

Opcode n-grams detect Android malware in [1]. Experimented on the frequencies of n-gram opcodes and succeeded in catching some families with perfect detection rates but failed for others. In the learning phase, used two classifiers, SVM and Random Forest (RF). Random forest achieved high accuracy of 96.88% with just bigrams. The metamorphic malware escaped the technique used, and there were fewer applications with that malware at the testing time. It worked well for polymorphic malware.

A comparison of static, dynamic, and hybrid analysis with the Hidden Markov Model (HMMs) machine learning technique done in [7]. API calls and opcode sequences are used as the features. In addition, 5-fold cross-validation is used for partition. Using dynamic for both training and testing got the best results for both the features when compared with entire static, training with static, and testing with hybrid and vice versa.

White box attacks include data poisoning and evasive attacks [9]. The author's will have knowledge of the model's specifics, training data, and relevant feature sets. On the other hand, the malware author will only know about the input features in the black box. We are working with the white box attacks in our project.

MADAM is a multi-level anomaly detector and a behavior-based detector designed to detect Android malware [10]. It extracts five groups of features from four different levels named kernel, application, user, and package. At the kernel level, system calls are monitored. The vast and sudden increase of the system calls detects misbehavior of the application. At the application level, both API calls and SMS are monitored. Most malware applications will have an SMS trojan to send SMS to the remote servers and capture the user contact list. Using SMS service will push financial costs on the user. The third level, user activity, is monitored on the user level. These level activities are related to the application level. Because the user or attacker will use the

SMS function. Finally, the package level monitors the static app metadata, mostly the permissions.

2.1.2 Adversarial Attacks on Android Malware

Explanation on how the adversary could evade static and dynamic Android malware detection techniques done in [8]. First, inject the benign opcode and benign system call sequences to avoid the malware detection mechanism. Then, opcode n-grams are used as features for static analysis, and system calls are used for dynamic analysis—used *TF-IDF* to inject the features of benign applications into the malware application. Only with a few injections do adversaries produce misclassification.

In [11], authors created twelve malware detection models and performed adversarial attacks. The assault is made to turn as many malware samples into adversarial samples as possible with the least amount of alteration to each sample. As a result, the proposed attacks has a good average fooling rate. Later, to defend against attacks made on detection models, authors introduced three adversarial defense tactics. First, the average accuracy was increased for permission-based and intent-based detection models using the suggested hybrid distillation-based protection method.

Single-policy attack was recommended in a white-box scenario when an adversary has complete knowledge of the detection mechanism [12]. They developed a reinforcement agent that uses a single Q-table technique to initiate an aggressive attack. The test findings demonstrated that with a few minor adjustments, a single-policy assault can successfully evade malware detection systems to achieve a high fooling rate. In addition, the authors developed a cutting-edge adversarial strategy called a multi policy assault for use in so-called "grey boxes," or situations when the attacker is unfamiliar with the model architecture and classification procedure [12]. The multi policy strategy produced the highest fooling rate, which is higher than that of the

single policy approach.

AdvAndMal, a system that uses a two-layer network for adversarial training to generate adversarial samples and improve the performance of classifiers in Android malware detection and classification [13]. The adversarial sample generation layer is composed of a conditional generative adversarial network called pix2pix, which can produce malware variants to increase the training set for the classifiers. The malware classification layer is trained using an RGB image visualized from a series of system calls. The overall framework's accuracy has raised.

Label flipping assaults have been proven to be successful at severely reducing system performance, even when the attacker's capabilities are limited. Label flipping are a specific type of data poisoning in which the attacker has control over the labels given to a subset of the training points. In [14], the authors suggested a way to identify and rename suspicious data points, reducing the impact of such poisoning assaults, as well as an effective algorithm to carry out optimal label flipping poisoning attacks.

The authors in [15] tested the performance and resilience of six machine learning algorithms against two distinct adversarial techniques using four different datasets. Label flipping at random and label flipping based on distance are the two options. Some machine learning methods show better robustness and performance results against adversarial attacks practically for all datasets.

CHAPTER 3

Methodology

We will explore various adversarial attack types and the attacks we use in our studies. A brief description of the machine learning and deep learning techniques we use in our research is also provided.

3.1 Adversarial Attacks

The objective of the adversarial attack is to increase the fooling rate by introducing clever perturbations to malware samples that drive misclassification and decrease the accuracy of the malware detection models. We learned that there are different adversarial attacks from [5]. We will briefly go over two types of attacks.

3.1.1 Data Poisoning Attack

A data poisoning attack is a causative attack. A causal assault contaminates data. Data corruption is its primary concern. It has different kinds of data corruption methods.

Label Flipping: By changing the training data's label from malicious to benign in this attack, the malware author corrupts the training data [16]. As a result, the detection system will be educated on this corrupted data, and during testing, it will classify the infection as benign. One of the methods we employ in our experiments is this one.

Injection of data: Even though the training data and malware detection model are unknown to the malware author, this approach works. The training set will merely receive fresh adversarial samples. The dataset will be tampered with in this manner, and the detecting system will malfunction.

Corruption of algorithm logic: The malware creator will be aware of the malware detection model and attempt to sabotage the algorithm's operational logic. This attack is brutal to build.

Although there are different kinds of data poisoning attacks, we will only use the label flipping attack that results from data poisoning.

3.1.2 Evasive Attack

Most people agree that the evasion attack is the most often utilized assault against malware detectors built using machine learning. An evasive attack is an exploratory attack [17]. The input data is purposefully changed to launch an evasion assault by tricking the machine learning system into classifying malware samples as benign.

White box attacks include data poisoning and evasive attacks [9]. The malware author's will have knowledge of the model's specifics, training data, and relevant feature sets. On the other hand, the malware author will only know about the input features in the black box. We are working with the white box attacks in our project.

3.2 Techniques for Classification

Classifying an input dataset involves predicting its class or label. Then, following the characteristics of the input data, the input dataset is mapped to the appropriate output class. The methods that we will employ to classify our experiments will be covered in this section. One deep learning approach, Multi-Layer Perceptron, and two machine learning techniques, Support Vector Machine, and Logistic regression are employed. The many machine-learning methods and their application to real-world problems are detailed in [18].

3.2.1 Multi-Layer Perceptron (MLP)

A neural network with numerous layers is a Multi-Layer Perceptron [19]. The outputs of some neurons become the inputs of other neurons when we join neurons to form a neural network. Generally speaking, there are three critical layers: the input layer, the hidden layers, and the output layer. MLP consists of one input layer, one or more hidden layers, and one output layer. Each input and output is separated

into nodes or neurons in the input and output layers. Every input will have a weight associated with it. The input and weights work together to determine whether or not a neuron should fire. The neuron will fire if the sum is higher than the threshold.

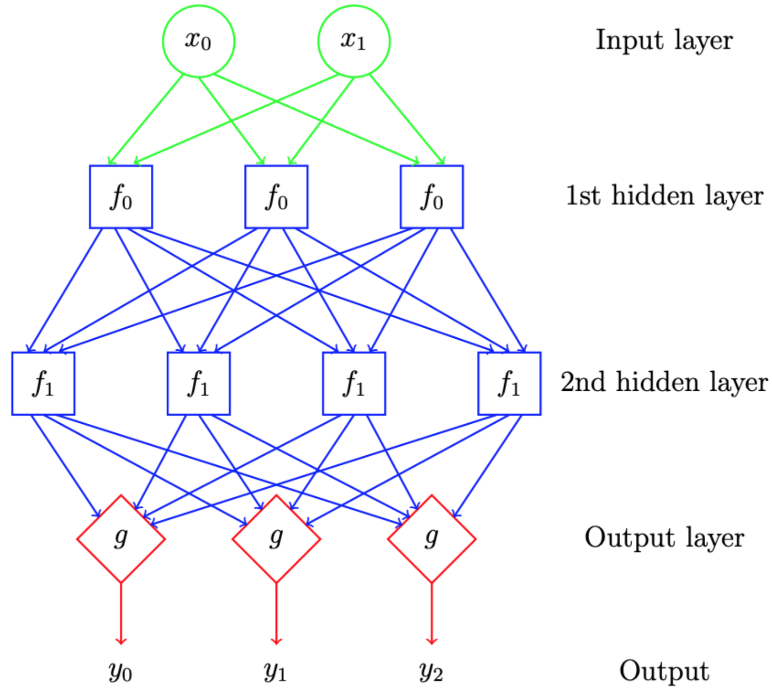

Figure 2: MLP with Two Hidden Layers [20]

In Figure 2, we can see a Multi-Layer Perceptron with two hidden layers. The weights of the MLP are completed after training and are assigned to each of its edges. The MLP can give some input features more weight than others and produce a decision boundary depending on which one categorizes the input features given the best. In [21], the author experimented with MLP for Android malware detection. The malicious applications that were disguised were found using the model. Therefore, we decided to use this technique in our experiments.

3.2.2 Logistic Regression (LR)

When the answer variable is categorical, the classification algorithm known as Logistic Regression is utilized. Logistic regression aims to find a correlation between

features and the likelihood of a specific outcome [22]. The classifier is a linear one in LR. It is a supervised learning technique for categorical data where some parameters depend on the input characteristics, and the output is a definite prediction. When a sigmoid function is fitted to the data in logistic regression, an S-shaped curve with values between 0 and 1 is produced. The sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

The input data are described, and a correlation is found using logistic regression. A dependent variable, a mean function to make predictions, and a link function that can change the mean function back into the distribution of the dependent variable are all needed for logistic regression. The underlying premise of logistic regression is that the independent variables are uncorrelated. Binomial logistic regression is logistic regression that fits two classes. A logistic regression model that includes more than two classes is referred to as multinomial logistic regression. This approach works best for our research because our trials use binary and multi-class data and are helpful for classification.

3.2.3 Support Vector Machine (SVM)

SVM's primary goal is to classify the dataset by maximizing the distance between the separating hyperplane and the dataset [18]. SVM is a supervised machine learning model used for classification. The tiny differences in malware samples from a particular family can be recognized using SVMs. In figure 3, we see the separating hyperplane, which maximizes the margin, and the data is linearly separable. The data points closest to the hyperplane are the support vectors. SVM uses of support vectors to increase the distance between the data points and the hyperplane.

SVM can be used for non-linearly separable data. Finding the ideal kernel can be difficult, but it can significantly increase classification accuracy with little to no

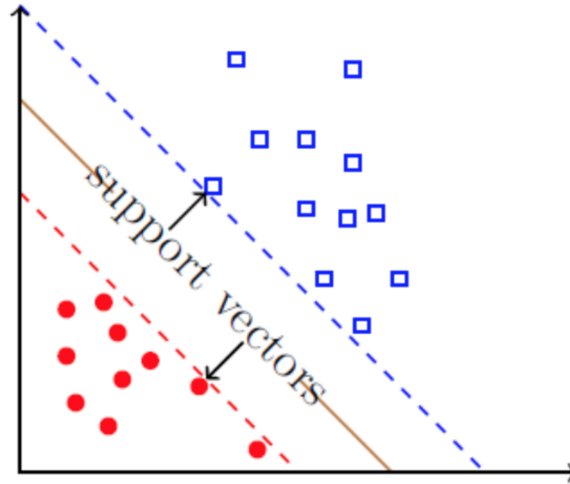

Figure 3: SVM with Separating Hyperplane [18]

additional computational burden. SVM has two methods to handle training data that cannot be separated linearly [18]. One approach is to choose a soft margin, which permits specific classification errors and creates a separable hyperplane. Another way is to map the input data to the feature space. The input data will be transformed into higher dimensions. In our project, we will be working with the linear SVM.

3.3 Metrics to Measure Performance

The performance of the models are assessed and examined using a variety of metrics, including Accuracy, Confusion Matrix, F1 Score, Recall, and Precision [23]. Accuracy measures how many forecasts our model correctly predicted. The number of occurrences between two, the true classification, and the anticipated classification are all recorded in the confusion matrix, a table. Before learning about precision and recall, there are a few things we need know.

1. True Positives (TP): These are the components that the model has designated as positive and genuinely are positive.

2. False Positives (FP): These are things that the model has classified as positive

but are actually negative.

3. False Negatives (FN): Elements are those that the model has classified as negative but which are actually positive.

4. True Negatives (TN): The projected and actual values are identical.

Precision gives an idea of what percentage of positive samples was accurate. In simple terms, our model indicates that the samples are positive, and they are. Recall shows what percentage of real positives were successfully identified. F1 Score is the precision and recall weighted average.

CHAPTER 4

Implementation

This chapter will briefly introduce our dataset, features, and malware types. Then, we will discuss how we are implementing the attacks. After that, we discuss how the models are built for our experiments.

4.1 Dataset

The dataset used in this research is KronoDroid [24]. It is a hybrid-featured dataset of Android malware that is appropriate for our goal. Figure 4 shows the dataset creation process in Kronodroid.

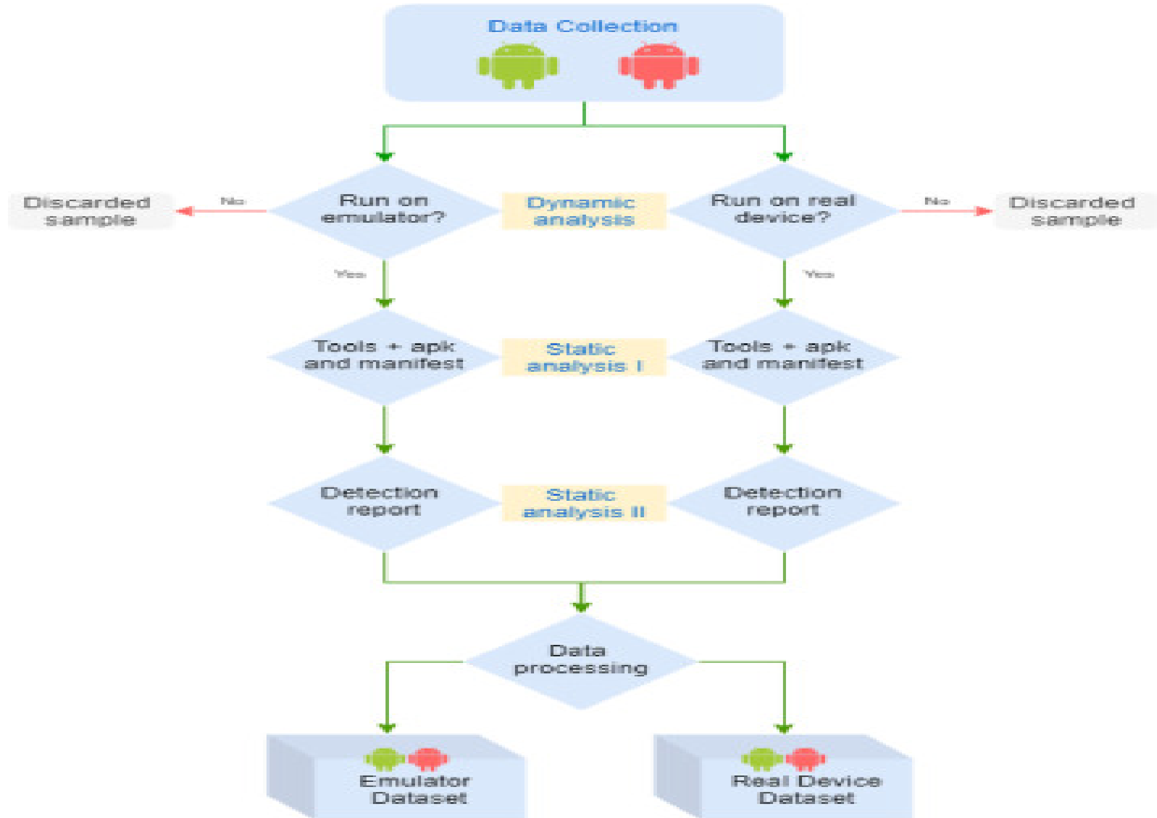

Figure 4: Process of Dataset Creation [24]

Both dynamic and static features can be found in this dataset. Real devices are also utilized to extract the functionalities in addition to the emulator. The dataset

has 78,137 samples, of which 41,382 are malware, and 36,755 are benign. Figure 5 shows the sample count where 0 represents benign and 1 represents malware. There are a total of 484 features available.

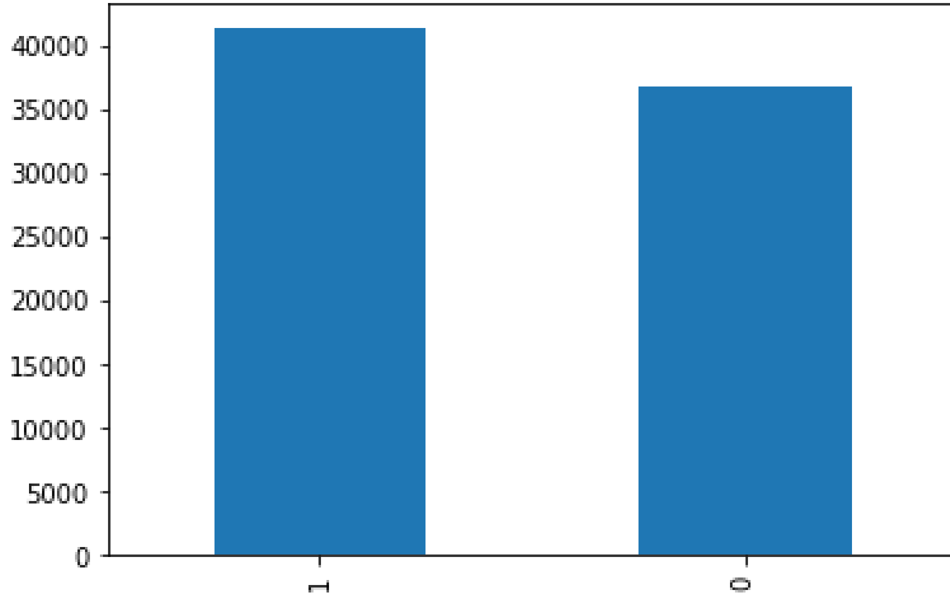

Figure 5: KronoDroid Samples Count

4.1.1 Data Preprocessing

We have to process the dataset, before we use them for our experiments. The caliber of the data used in machine learning implementations is one of the most crucial duties. Outliers, Missing data etc. can all have a big impact on how well a model performs. Utilizing the data as-is had a negative impact on model performance and produced low accuracy. We removed 36 duplicate samples in the dataset, which makes 78,101 samples.

4.1.1.1 Data for binary classification to perform Label Flip

From 484 features, five are being removed. The features eliminated include "Package," "MalFamily," "sha256," "EarliestModDate," and "HighestModDate." The terms "MalFamily," "Package," and "sha256" will make it crystal evident if something

is malicious or not. "MalFamily" will be used for multi-class classification.

4.1.1.2 Data for multi-class classification to perform Label Flip

The malware samples in this dataset represent 240 malware families. We are considering the top 9 malware families for our multi-class classification, as shown in Figure 6. These malware families correspond to numerous malware types, including trojans, worms, and viruses. Here is a brief description of each family.

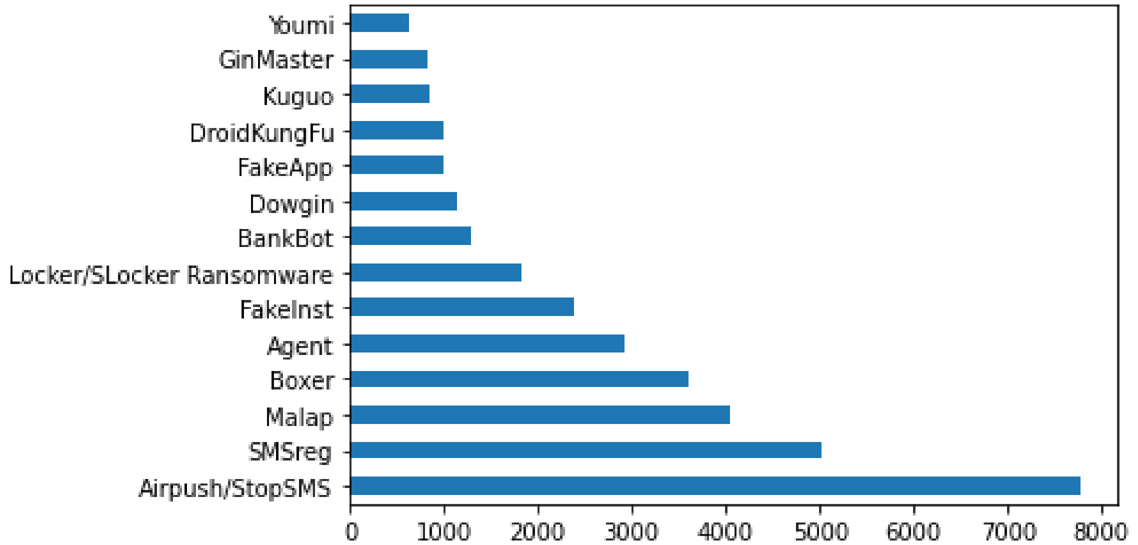

Figure 6: KronoDroid Top Families

- `Airpush/StopSMS`: An aggressive advertising network is Airpush [25]. The SMS message containing the download Uniform Resource Locator (URL) serves as the malware's primary spread method.
- `SMSreg`: The battery utilization of a device is said to be maximized by SMSreg, which is offered under the name "Battery Improve" [26]. Without the user's knowledge or approval, it secretly gathers data from the device.
- `Malap`: Another form of information stealing malware.
- `Boxer`: A family of malware known as Boxer makes money via sending SMS messages. It poses as a trustworthy installer or application downloader, but it

secretly sends SMS messages in the background once set up. The users' account is billed for costs related to its SMS sending actions [25].

- `Agent:` Unknown to the user, the malicious program agent runs in the background of a mobile device. It waits silently for orders from a Command and Control (C&C) server [27]. These instructions could be anything from stealing and transferring sensitive data to distant sites to acting as Distributed Denial of Service (DDoS) bots against specific targets.
- `FakeInst:` This particular family of malware simulates the installation of a legitimate app by displaying a false window to the user. In addition, the malware discreetly sends SMS messages to premium-rate short numbers during the false installation [25].
- `Locker/SLocker Ransomware:` Locker/SLocker Ransomware is purportedly the first file-encrypting Android ransomware. It is also notable for communicating with its controller through the network.
- `BankBot:` The purpose of BankBot is to steal payment and banking credentials from the user's mobile [28]. In addition, it deceives users into providing their bank information by displaying an overlay window that looks exactly like the bank mobile app login page.
- `Dowgin:` Dowgin is an advertisement app that is typically distributed as part of a package with other programs that are trustworthy [25]. The advertising app is used to display ads; while doing so, it may also discreetly collect and send information from the device.

After taking into account 9 families, we obtained 30,017 samples in total. Table 1 shows the number of samples in each family. In terms of features, we employ the same ones as binary classification with the exception of "MalFamily". In our multi-class

system, this feature serves as the label.

Table 1: Families Count in KronoDroid

Family name	Sample count
Airpush/StopSMS	7775
SMSreg	5019
Malap	4055
Boxer	3597
Agent	2930
FakeInst	2384
Locker/SLocker Ransomware	1815
BankBot	1297
Dowgin	1145

4.1.1.3 Data for binary classification to perform evasive attack

We must lower the number of features to undertake the evasive assault on the binary classification. Our data collection contains 478 features. Using Recursive Feature Elimination with Cross Validation (RFECV) and logistic regression, features were reduced to 28. The features which we are using for experiment are mentioned in Table 2

4.2 Implementation of Adversarial Attacks

We provide a detailed explanation of the attacks we use on our dataset and model. We are developing two attack strategies. Label flipping is one, while evasive attack is another. In all our experiments, we are dividing the dataset into 25% for testing and 75% for training.

4.2.1 Label Flip on Binary Classification

There are other approaches to flip the labels, including using the same samples, selecting them at random, or using certain criteria. We are working on randomly converting the labels in the train dataset. Two labels are available in a binary classification problem: malicious(1) or benign (0). Once we divided our dataset, we

Table 2: Selected Features to Perform Evasive Attack

Features selected
sigaltstack
fchmod
truncate
fstatfs64
getsockopt
sysinfo
wait4
getrlimit
SYS_306
SYS_312
SYS_333
SYS_339
ACCESS_COARSE_LOCATION
ACCESS_FINE_LOCATION
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
BIND_QUICK_SETTINGS_TILE
BODY_SENSORS
BROADCAST_STICKY
FOREGROUND_SERVICE
MEDIA_CONTENT_CONTROL
MOUNT_UNMOUNT_FILESYSTEMS
READ_CONTACTS
RECEIVE_BOOT_COMPLETED
RECEIVE_MMS
SYSTEM_ALERT_WINDOW
dangerous
Detection_Ratio

worked on various sample flipping percentages. We worked with a proportion of 10, 20, 30, 40, 50, 60, and 70. Therefore, we will flip that many percentages of the labels from the training sample for each %. We reverse the label to 0 (benign) if it is 1 (malware) and vice versa. If the sample has already been flipped, we move on to the other sample for flipping.

4.2.2 Label Flip on Multi-Class Classification

We have multi-class labels in this case, as the name says. We will have 9 class labels because we are taking into account the top 9 families. This classification divides the dataset into 75% for training and 25% for testing. The class label flipping requirement requires that a class label be chosen randomly from the remaining class labels. If the class label has already been modified, we shall move on to the following sample and skip that label. Additionally, the sample will be chosen at random. In this classification, we also use the exact sample corruption percentages as binary classification.

4.2.3 Evasive Attack on Binary Classification

Instead of using training samples in this attack, we use testing samples. We use the linear SVM on the training data to locate the benign sample closest to the hyperplane. We might claim that this sample represents the malware sample that has undergone the fewest changes possible to become as innocent. We identify the characteristics that need to change to make malware more benign. To compare the error rate, we will start with a testing sample corruption rate of 10% and then increase it by 10% each time.

4.3 Implementation of Techniques

SVM, LR, and MLP were the three algorithms we chose to use. The grid search technique selects the optimal parameters for each model before production. A method for locating the ideal parameter values in a grid from a set of parameters is called GridSearchCV. In essence, it is a cross-validation method. Predictions are performed after extracting the ideal parameter values. Finally, we will go into detail regarding model hyper parameters, building and training.

CHAPTER 5

Results

For both attacks listed above, we received findings as graphs, and the following sections contain an in-depth discussion of those results.

5.1 Label Flip Attack on Binary Classification

Before attacking the model, we have to build the model. GridSearchCV selects the best parameters for the models. The parameter selection for all the three models for binary classification on Label flip attack can be seen in Table 3. Without making any attacks, compare the accuracies of all three models in Figure 7. The accuracy of the Multi-Layer Perceptron is slightly higher than that of the other two models.

Table 3: Hyperparameters for Label Flip Attack on Binary Classification

Model	**Hyperparameters**	**Tested values**	**Accuracy**	
			Train	**Test**
SVM	*C* gamma	(**0.1**,1,10) (0.001,**0.01**,0.1,1,10,100)	0.992	0.990
LR	*C* solver penalty	(**0.1**,1,10) (newton-cg, **lbfgs**) (**l2**,l1)	0.992	0.992
MLP	solver max_iter hidden_layer_sizes activation alpha early_stopping	(**adam**, sgd) (100, **1000**) ((200),(300),**(400)**,(100,100),(200, 200)) (**tanh**, relu) (0.0001,0.001,**0.005**) (True, **False**)	0.998	0.994

5.1.1 Support Vector Machine

We utilized a linear kernel, C as 0.1 and *gamma* as 0.01, for the SVM. SVM achieved a 99% accuracy rate with these ideal settings. We now apply the attack to the model we choose. Let us start by using a 10% corruption to the training samples, which is 5,857 samples out of 58,575 samples. Our accuracy result was 98.7%, which indicates a decline. Next, try flipping 20% of the training samples which is 11,715

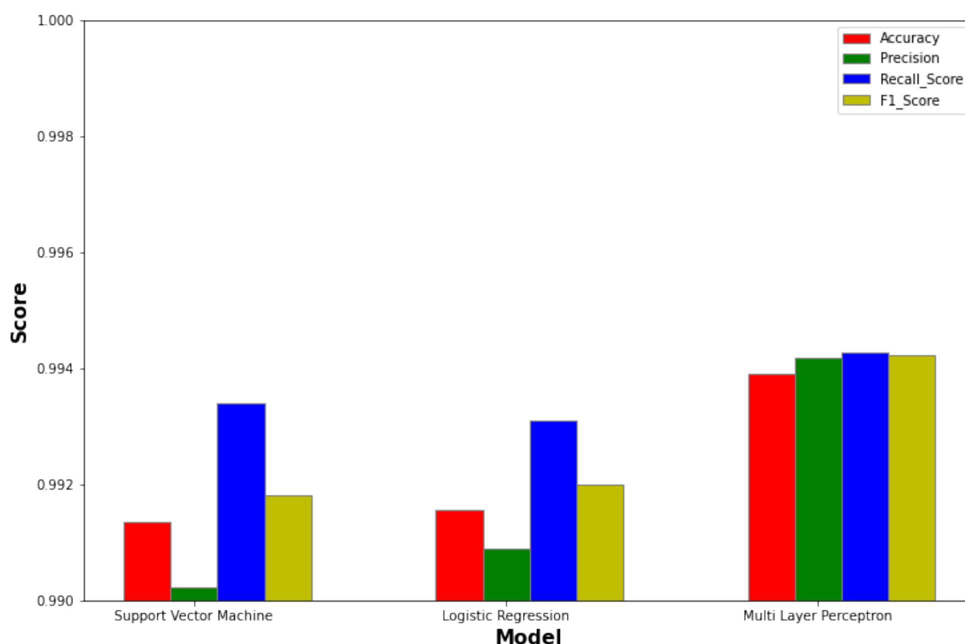

Figure 7: Accuracy of Binary Models without any Attack

samples. A 98.5 percent accuracy was attained. This time, there is not much of a decrease.

Currently, 17,572 out of 58,575 training samples which is 30%, are being flipped. We have not noticed much decline in accuracy as of yet. We got 0.983. After 40% of flipping, we see a sudden decrease in the accuracy. Similarly, we experimented with different corruption levels in the training data. We can observe the training and testing accuracy for the various percentages of sample corruption in Figure 8 scores. The testing score visibly decreases when we increase the training sample corruption rate.

5.1.2 Logistic Regression

The ideal values for C, $solver$, and $penalty$ for LR with grid search are "$lbfgs$," "$l2$," and "0.1." This model got a 99.1% accuracy rate. Execute the assault against the model. First, we corrupt 10% of the training samples and still managed to achieve

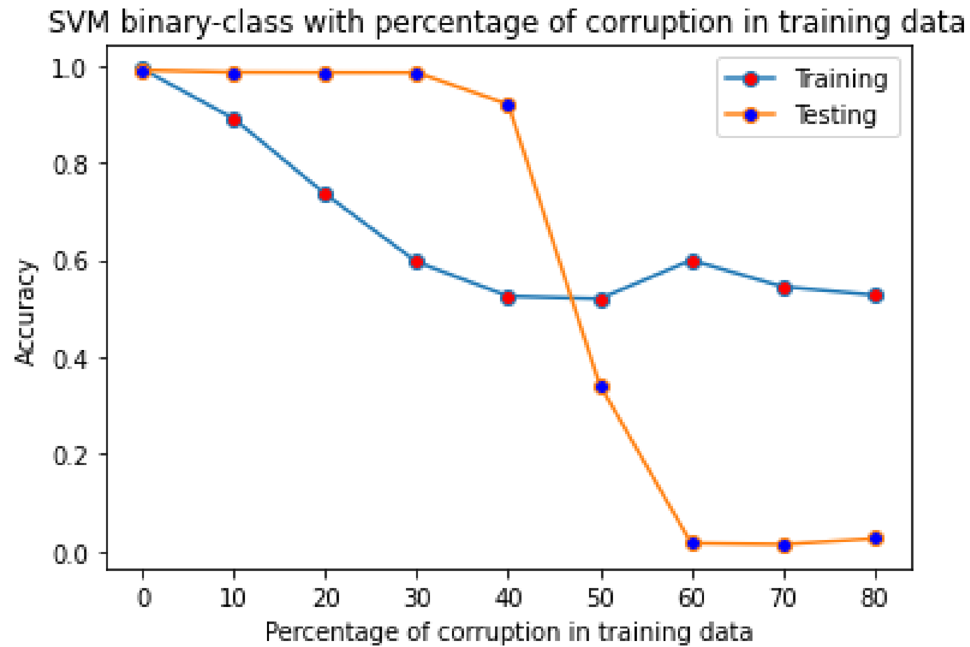

Figure 8: SVM Binary Classification with Label Flip Attack

an accuracy of 98.9%, matching the SVM's results. There has been no noticeable decline in accuracy after flipping the labels of 20% of the samples, which resulted in an accuracy of 98.7%.

By inverting their labels, try again with a 30% sample corruption. A 97.8% accuracy rate was attained. Following, we observe a further accuracy reduction. We calculated the remaining sample corruption percentage and displayed the findings in Figure 9. We can observe some accuracy that is somewhat similar to or close to SVM in Figure 11.

5.1.3 Multi-Layer Perceptron

We employed a Multi-Layer Perceptron with a single hidden layer of 400 neurons. The activation function is "$tanh$," and the maximum number of iterations is 1000. Our accuracy percentage of 99.4% is more significant than what SVM and LR obtained. We are now going to tackle the model. Start with a training set that has 10% sample

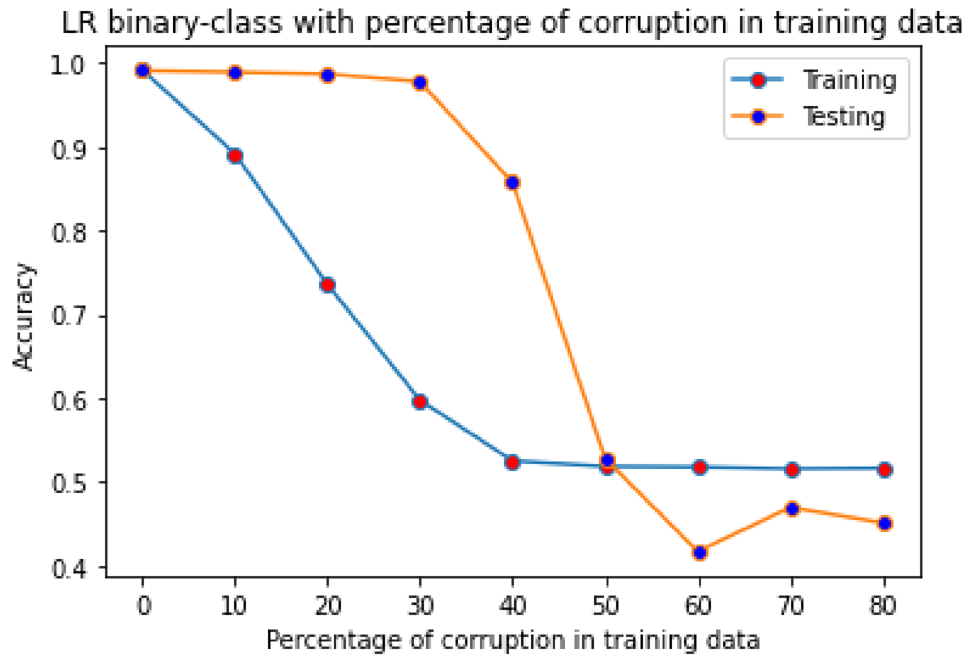

Figure 9: LR Binary Classification with Label Flip Attack

corruption with the attack. Obtained a 95% accuracy rate, which is lower than the results of the two classical machine learning methods.

20 percent of the training samples were corrupted by classifying malware as benign and benign as malware. We got an accuracy of 82%, which is a fairly sharp decline in accuracy. We attempted to increase the amount of corruption in the samples, and the results are shown in Figure 10.

We may evaluate the three models accuracy under a label flip attack with varying percentages of obfuscated samples in Table 4. Although the traditional approaches are steady for the first few runs, we later observe accuracy loss along with MLP. While in MLP, the decrease is visible even in the initial experiments. We can conclude from this series of experiments that the traditional models performed better in Figure 11.

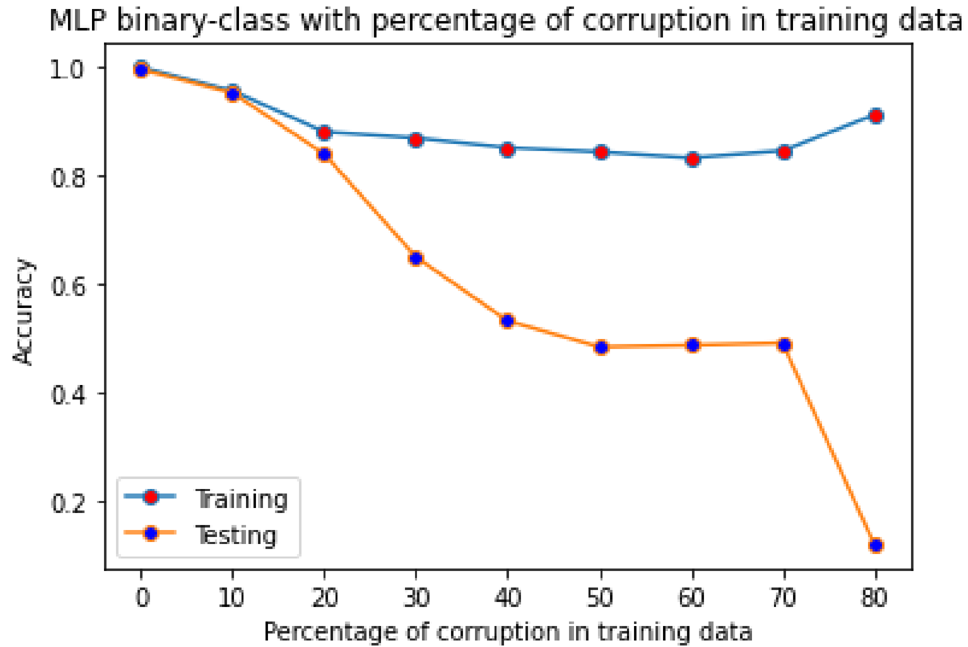

Figure 10: MLP Binary Classification with Label Flip Attack

Table 4: Comparison of Accuracy with Label Flip Attack for Binary

Model	0%	10%	20%	30%	40 %	50 %	60%
SVM	0.990	0.986	0.985	0.985	0.919	0.342	0.016
LR	0.991	0.989	0.987	0.979	0.857	0.529	0.417
MLP	0.994	0.951	0.838	0.648	0.530	0.482	0.485

5.2 Label Flip Attack on Multi-Class Classification

We must first construct the model before assaulting it. The ideal model parameters are chosen for the models by GridSearchCV as shown in Table 5. Compare the accuracies of all three models in Figure 12 without making any assaults. We can see Multi-Layer Perceptron model performing better than the remaining classic models SVM and LR.

5.2.1 Support Vector Machine

For the SVM model, we used a linear kernel with C as 10 and *gamma* as "*auto*". With these optimum settings, SVM had a 92.5 percent accuracy rate. We can view

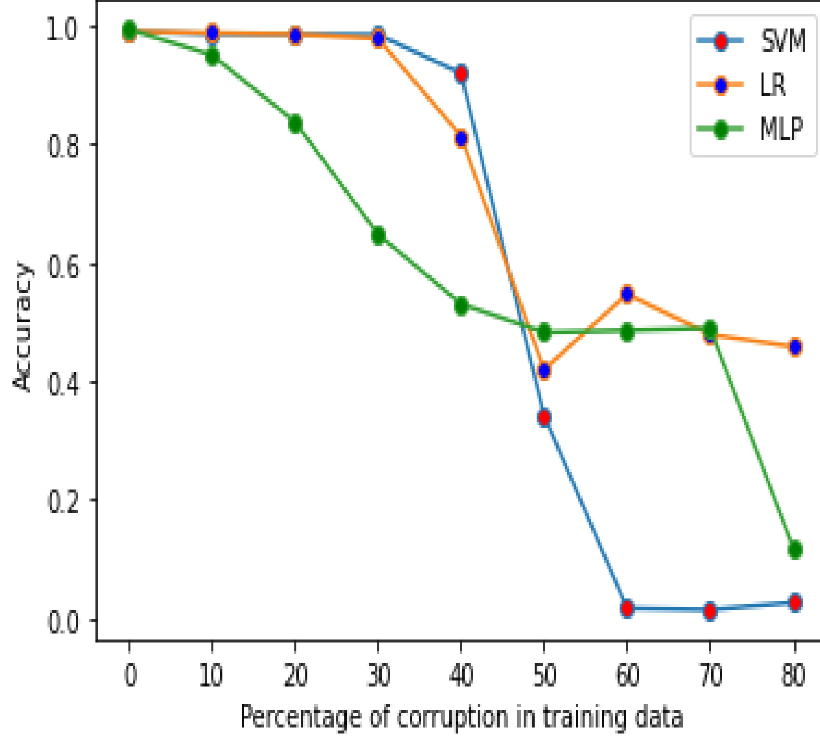

Figure 11: Label Flip Attack on the Models

Table 5: Hyperparameters for Label Flip Attack on Multi-Class Classification

Model	Hyperparameters	Tested values	Accuracy Train	Accuracy Test
SVM	C gamma	(0.1,1,**10**) **auto**	0.925	0.910
LR	C solver penalty	(0.1,**1**,10) (newton-cg, **lbfgs**) (l2, l1, **none**))	0.915	0.900
MLP	solver max_iter hidden_layer_sizes activation alpha early_stopping	(**adam**, sgd) (100,1000,**10000**) ((200),(300),(400),(100,100),**(200, 200)**) (logistic, tanh, **relu**) (0.0001,0.001,**0.005**) (True, **False**)	0.994	0.926

the recall, precision, and f1-scores for all the families prior to the attack in Table 6.

Now, we apply the label flip attack on our model. In this approach, we switch

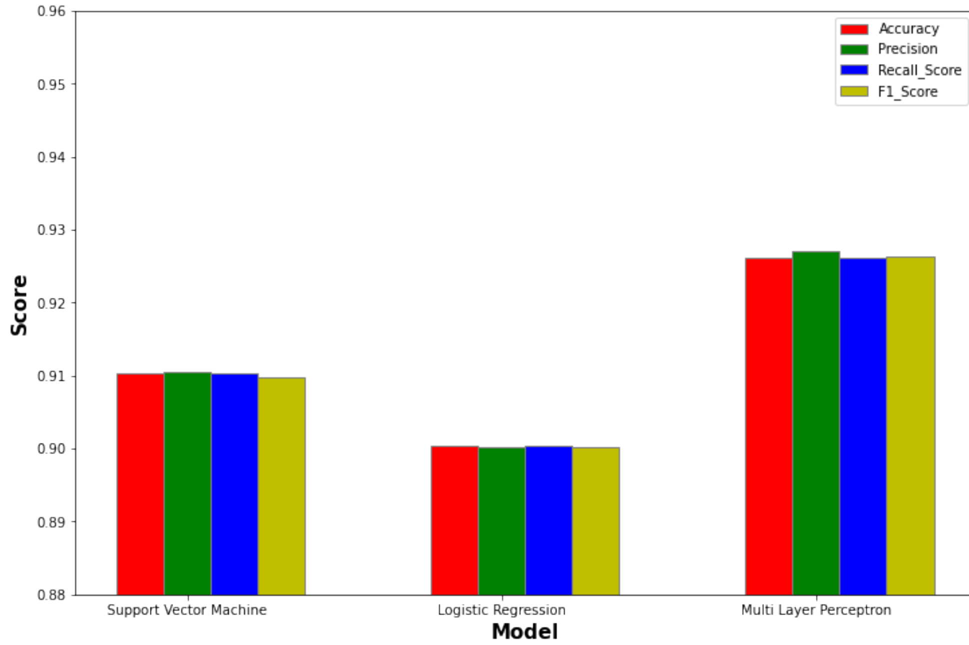

Figure 12: Accuracy of Multi-Class Models without any Attack

Table 6: Recall, Precision and F1-score before Attack with SVM

Family	Recall	Precision	F1-score
Airpush/StopSMS	0.97	0.98	0.98
SMSreg	0.92	0.93	0.93
Malap	0.91	0.87	0.89
Boxer	0.96	0.99	0.97
Agent	0.73	0.80	0.76
FakeInst	0.84	0.84	0.84
Locker/SLocker Ransomware	0.97	0.88	0.92
BankBot	0.95	0.98	0.97
Dowgin	0.76	0.60	0.67

the sample's label to a different class that is been chosen at random. For instance, if the sample label we chose is 2, we will swap it out for another label that is chosen at random from 1 to 9 and not 2. Starting with 2,250 samples out of 22,512 training samples, let us apply a 10 percent corruption to the samples. The accuracy result we obtained was 89.5%, which is declining. The recall, precision, and f1-scores after 10

percent attack can be seen in Table 7.

Table 7: Recall, Precision and F1-score after 10% Attack with SVM

Family	Recall	Precision	F1-score
Airpush/StopSMS	0.96	0.97	0.97
SMSreg	0.91	0.92	0.92
Malap	0.88	0.87	0.88
Boxer	0.96	0.98	0.97
Agent	0.72	0.78	0.75
FakeInst	0.87	0.80	0.83
Locker/SLocker Ransomware	0.95	0.87	0.91
BankBot	0.91	0.98	0.94
Dowgin	0.72	0.58	0.64

Next, we attempted flipping the labels for many classes with 20% of the training sample corrupted. With an accuracy of 87.6%, we saw a fairly sharp decline in accuracy. We tried flipping 30% of randomly selected samples with various class labels and obtained an accuracy of 84.1%, which is not a significant change. In the training data, we also experimented with various amounts of corruption with 10 percent of increase each time till 80 percent. In Figure 13, we can see the testing accuracy for the different sample corruption percentages. The testing score visibly decreases as the training sample corruption rate rises. There is a noticeable decline after 30 percent sample tampering.

5.2.2 Logistic Regression

For LR with grid search, the best values for *solver*, and *penalty* are "$lbfgs$," "*none*" respectively. This model's accuracy rating was 90%. Different scores of the model for different families can be seen in Table 8. Carry out the attack on the model. This model also uses the same attack that we used on the SVM. First, we were able to corrupt 10% of the training samples and get an accuracy of 88.1%. Accuracy suffers when there is even a 10% contamination. The recall, precision and f1-scores before

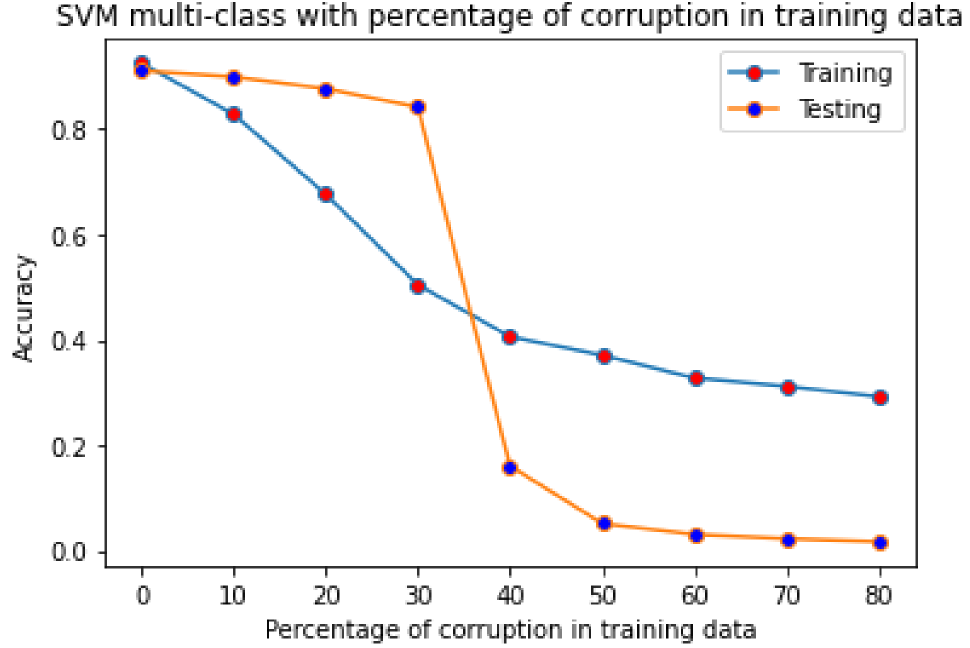

Figure 13: SVM Multi-Class Classification with Label Flip Attack

the attack are listed in the Table 9.

Table 8: Recall, Precision and F1-score before Attack with LR

Family	Recall	Precision	F1-score
Airpush/StopSMS	0.97	0.97	0.97
SMSreg	0.92	0.93	0.92
Malap	0.89	0.85	0.87
Boxer	0.96	0.99	0.98
Agent	0.73	0.75	0.74
FakeInst	0.83	0.83	0.83
Locker/SLocker Ransomware	0.93	0.89	0.91
BankBot	0.95	0.98	0.96
Dowgin	0.67	0.65	0.66

20% of the training instances had labels that we tried flipping to a new class. Obtained an accuracy of 85.7%. The remaining sample corruption percentage was explored with. With an increase in training sample corruption, the testing score degrades clearly. Figure 14 makes it crystal evident how accuracy decreases as the

Table 9: Recall, Precision and F1-score after 10% Attack with LR

Family	Recall	Precision	F1-score
Airpush/StopSMS	0.95	0.97	0.96
SMSreg	0.89	0.90	0.90
Malap	0.88	0.85	0.86
Boxer	0.94	0.98	0.96
Agent	0.71	0.72	0.71
FakeInst	0.81	0.74	0.77
Locker/SLocker Ransomware	0.90	0.88	0.89
BankBot	0.91	0.98	0.94
Dowgin	0.65	0.61	0.63

percentage of distorted samples rises.

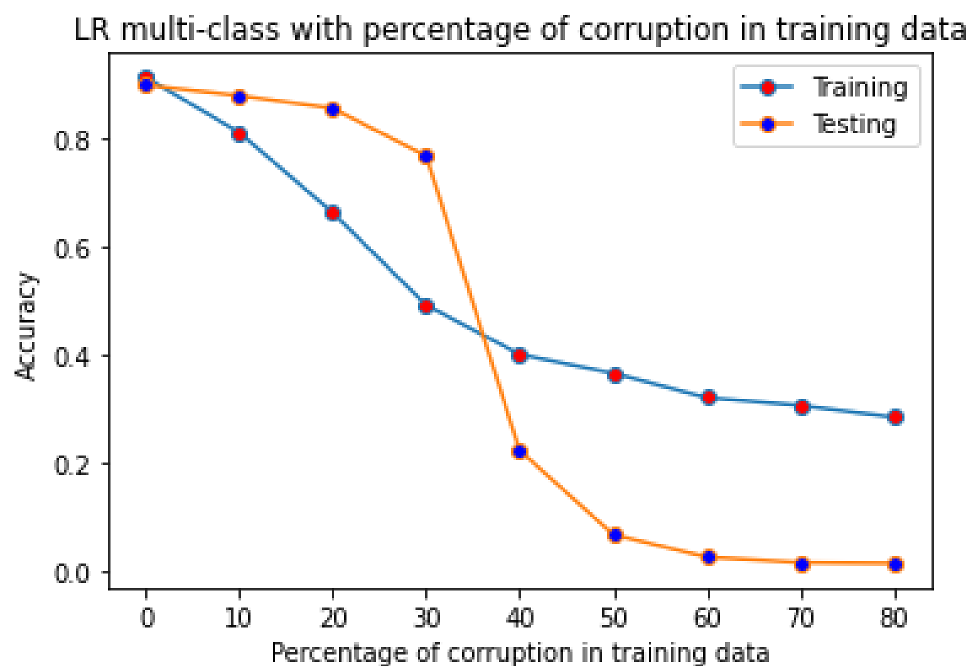

Figure 14: LR Multi-Class Classification with Label Flip Attack

5.2.3 Multi-Layer Perceptron

We employed a Multi-Layer Perceptron with a two hidden layers of 200 neurons each. The activation function is "*relu*," and the maximum number of iterations is 10000. Our accuracy percentage of 92.6% is more significant than what SVM and LR

obtained. In Table 10, we can see the different scores of all the families. We are now

Table 10: Recall, Precision and F1-score before Attack with MLP

Family	Recall	Precision	F1-score
Airpush/StopSMS	0.98	0.97	0.98
SMSreg	0.95	0.93	0.94
Malap	0.93	0.93	0.93
Boxer	0.98	0.99	0.98
Agent	0.74	0.80	0.77
FakeInst	0.87	0.89	0.88
Locker/SLocker Ransomware	0.96	0.93	0.94
BankBot	0.97	0.97	0.97
Dowgin	0.80	0.66	0.72

going to tackle the model. Start with a training set that has 10% sample corruption with the attack. Obtained a 89.8% accuracy rate, which is higher than the results of the two traditional machine learning techniques. The results of the model after the attack is displayed in Table 11. Next, tried with 20% samples flipping in the training set and got an accuracy of 79.4%. We attempted to increase the amount of corruption in the samples, and the results are shown in Figure 15.

Table 11: Recall, Precision and F1-score after 10% Attack with MLP

Family	Recall	Precision	F1-score
Airpush/StopSMS	0.97	0.91	0.94
SMSreg	0.86	0.93	0.89
Malap	0.91	0.92	0.91
Boxer	0.96	0.98	0.97
Agent	0.77	0.73	0.75
FakeInst	0.82	0.87	0.84
Locker/SLocker Ransomware	0.92	0.90	0.91
BankBot	0.94	0.97	0.96
Dowgin	0.73	0.72	0.73

We have performed all the experiments on the three different models with different percentage of corruption with label flip attack and the results are shown in Figure 16.

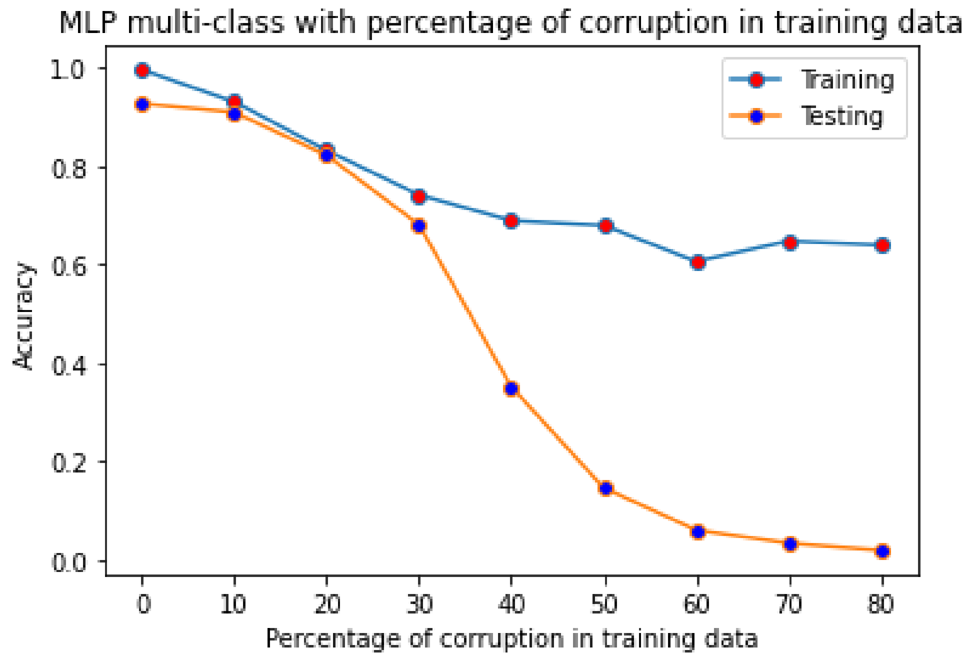

Figure 15: MLP Multi-Class Classification with Label Flip Attack

We may evaluate the three models accuracy under a label flip attack with varying

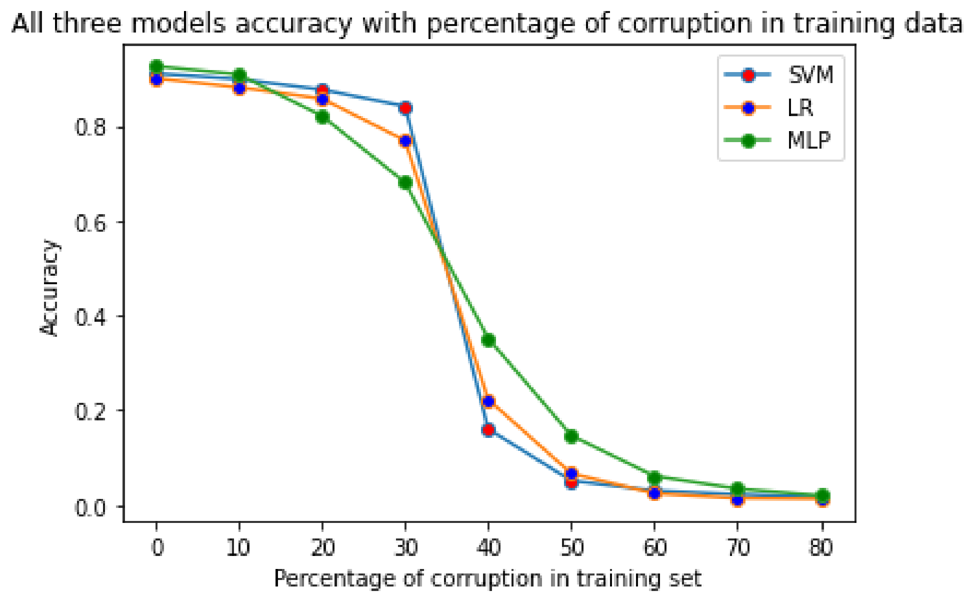

Figure 16: Label Flip Attack on the Multi-Class Models

percentages of obfuscated samples in Table 12. Although the MLP is steady for

the first few runs, we later observe accuracy loss along with SVM and LR. We can conclude from this series of experiments that the SVM model performed little better than other two models.

Table 12: Comparison of Accuracy with Label Flip Attack for Multi-Class

Model	0%	10%	20%	30%	40 %	50 %	60%
SVM	0.910	0.895	0.876	0.841	0.160	0.051	0.031
LR	0.900	0.881	0.857	0.769	0.223	0.067	0.025
MLP	0.926	0.898	0.794	0.636	0.334	0.144	0.075

5.3 Evasive Attack on Binary Classification

Before we initiate the assault, we must first determine the features that must be changed to convert malicious software samples into benign ones. The dataset consists of 78,137 samples and 28 attributes. Using the linear SVM, the benign point closest to the hyperplane is identified. Identify the qualities that need to be modified next. We can see the feature importance of all the 28 can be seen in Figure 17. We are considering the positive ones which are 17. Table 13 provides a list of the 17 essential characteristics that must be changed for the malware sample in order for the model to recognize it as benign.

Before using an evasive assault on any of the three models, the accuracy of each is shown in Figure 18. Comparing the MLP to the other two models, it is more accurate. For this experiment, we did not continually train the model because the training data did not vary, as we did with the label flip attack. It only changes the malicious sample in test data. In Table 14, we can see the model best parameters chosen for each model.

5.3.1 Support Vector Machine

Using GridSearchCV, we identified the most productive experimental parameters. There are only 28 features to consider in this situation. A linear SVM's optimal C

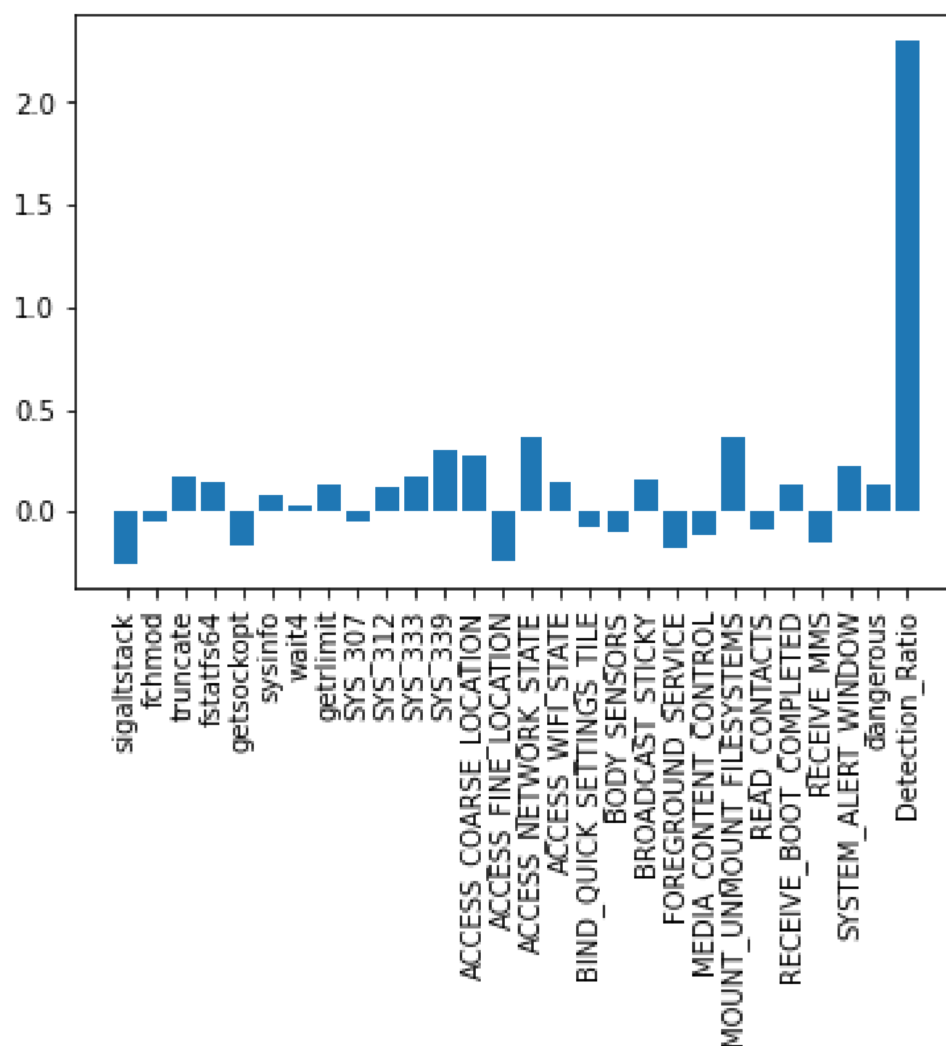

Figure 17: Feature Importance for Selected 28 Features

and *gamma* values are 0.1 and 0.01, respectively. With these settings, we trained the model, and the result was an accuracy of 99.1%, which is a respectable result. It's time to start attacking the trained model now. By replacing the values for the 17 features listed in Table 13 with values obtained from the benign point values which is near to hyperplane, we taint the testing sample. Start by corrupting 10% of the samples. From 19,526 testing samples, we corrupt 1,952 samples.

Only 10% of the test sample contamination resulted in an accuracy of 94.6%. Try

Table 13: 17 Important Features to Make Malware as Benign

Features
truncate
fstatfs64
sysinfo
wait4
getrlimit
SYS_312
SYS_333
SYS_339
ACCESS_COARSE_LOCATION
ACCESS_NETWORK_STATE
ACCESS_WIFI_STATE
BROADCAST_STICKY
MOUNT_UNMOUNT_FILESYSTEMS
RECEIVE_BOOT_COMPLETED
SYSTEM_ALERT_WINDOW
dangerous
Detection_Ratio

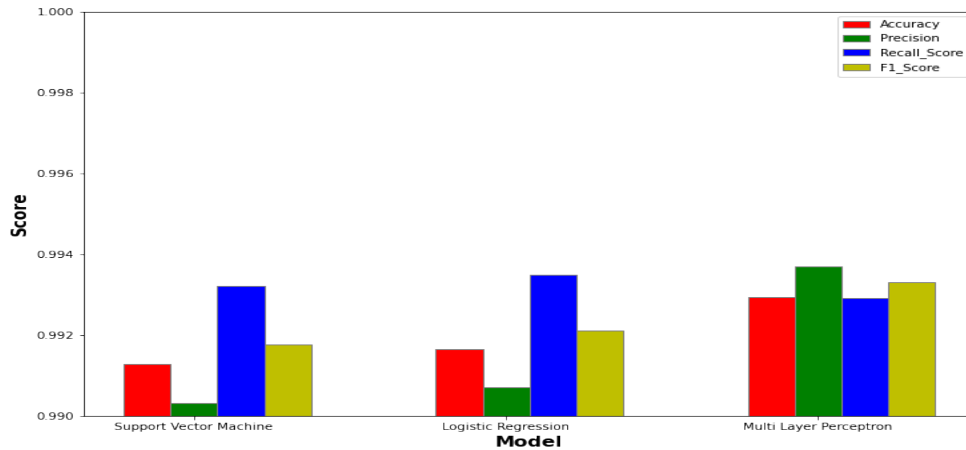

Figure 18: Accuracy of Models before Evasive Attack

a 20% test sample now, paying specific attention to the malware samples. A 89.8% accuracy rate was obtained. We can see that the accuracy decreases by 5% every time the corruption goes up by 10%. The training and testing accuracy of the model for each 10% till 60% of obfuscation is shown in Figure 19. As we previously stated, the

Table 14: Hyperparameters for Evasion Attack on Binary Classification

Model	**Hyperparameters**	**Tested values**	**Accuracy**	
			Train	**Test**
SVM	C gamma	(**0.1**,1,10) (0.001,**0.01**,0.1,1,10,100)	0.990	0.991
LR	C solver penalty	(**0.1**,1,10) (newton-cg, **lbfgs**) (**l2**, l1))	0.991	0.992
MLP	solver max_iter hidden_layer_sizes activation alpha early_stopping	(**adam**, sgd) (100,1000,**10000**) ((200),(300),(400),(100,100),**(200, 200)**) (logistic, tanh, **relu**) (0.0001,0.001,**0.005**) (True, **False**)	0.996	0.993

training score would remain unchanged because we did not alter the training samples.

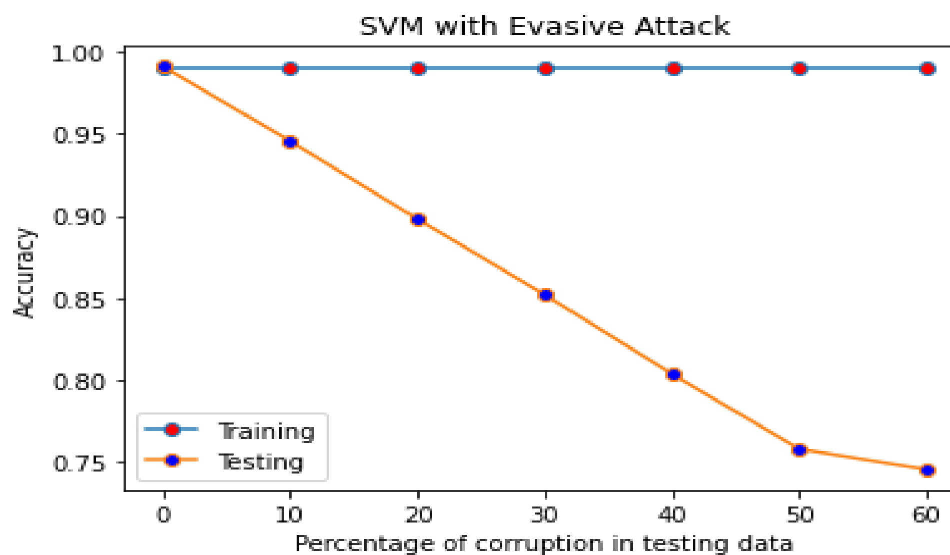

Figure 19: SVM with Evasive Attack

5.3.2 Logistic Regression

We identify the model's ideal parameters as C as 0.1, *penalty* as "$l2$", and *solver* as "$lbfgs$". With these settings, we created a logistic regression model with a 99.1% accuracy rate, which is slightly equal to SVM. As soon as our model is prepared, we launch an evasive attack against it. Now, work with 10% of the test samples that

have features taken from the training sample's benign sample. When we apply a 10 percent corruption, our accuracy dropped by 6%, giving us a result of 93.06 percent.

We performed a 20 percent corrupted sample injection into the test data and obtained an accuracy of 86.9 percent. We run tests using the remaining attack percentage. In Figure 20, we can see the accuracy versus corruption ratio. As the corruption percentage rises, accuracy decreases linearly in the opposite direction. The training score is same for all the experiments as they is no change in the training samples.

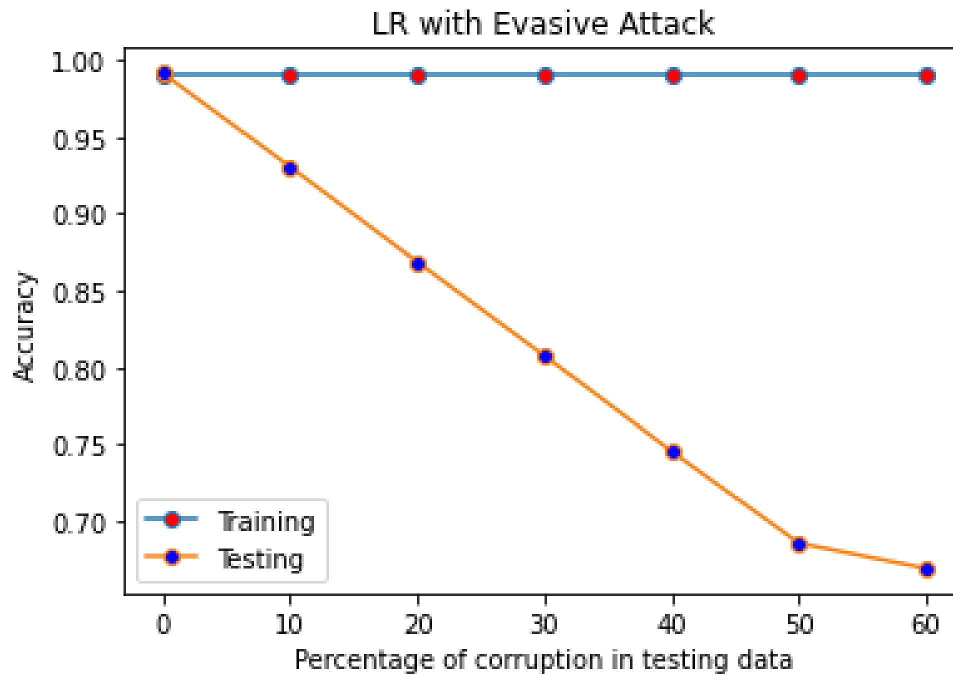

Figure 20: LR with Evasive Attack

5.3.3 Multi-Layer Perceptron

First, using GridSearchCV's top parameter picks, we build the model. These are the best parameters we could come up with, with a 99.3% accuracy rate, and they include the activation function "$relu$," two hidden layers with 200 neurons each, and a maximum iteration of 10,000. The base accuracy of the other two models is roughly

comparable.In the same way that we did for the other two models, conceal 10% test samples with the attributes we obtained from the benign sample that is closest to the hyperplane. The accuracy of the test using the base model is now 96.9%.

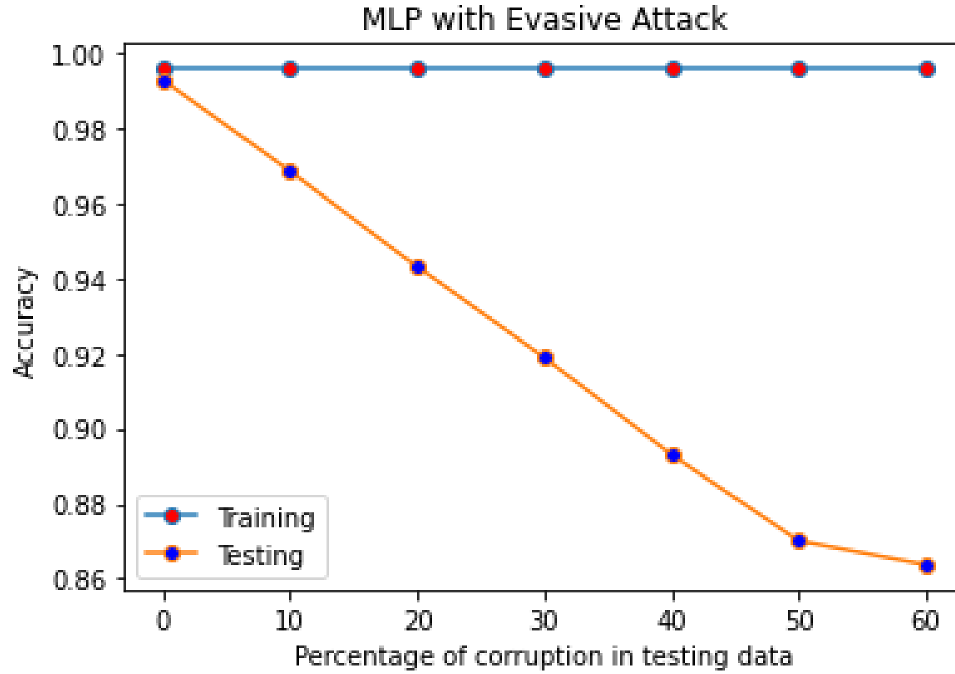

Figure 21: MLP with Evasive Attack

Now, we run the test using samples that have 20% corruption. Our accuracy was 94.3%. We work with percentages of 30, 40, 50, and 60. Because there are roughly 60% malware samples in the test set, we are unable to get above this level. We can view the results of the trials with the remaining percentages in the Figure 21. Every time, we observe a 2 percent accuracy loss for a 10 percent rise in corruption.

Finally, we can observe the correctness of each of the three models along with the proportion of corruption in Figure 22. Comparing Multi-Layer Perceptron to the other two models, we can say that it did well in this evasive attack. We can see the accuracy of each of the three models at each level of corruption in Table 15.

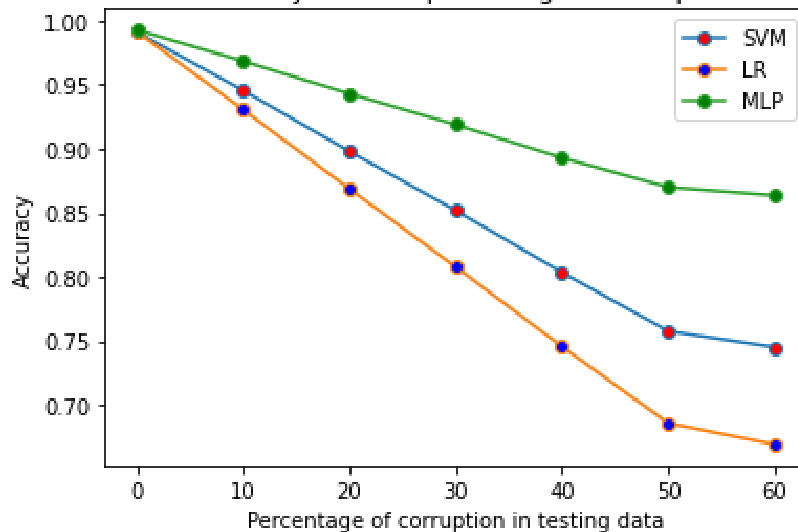

Figure 22: Evasive Attack on the Models

Table 15: Comparison of Accuracy with Evasive Attack

Model	0%	10%	20%	30%	40 %	50 %	60%
SVM	0.991	0.946	0.898	0.852	0.804	0.758	0.745
LR	0.992	0.931	0.869	0.807	0.746	0.686	0.669
MLP	0.993	0.969	0.943	0.919	0.893	0.870	0.864

CHAPTER 6

Conclusions and Future Work

This chapter examines the project's end and potential follow-up tasks. There is a lot of study being done related to the adversarial attacks. This project can be expanded using a wide variety of additional methods and strategies.

We employed two different adversarial attack methods in this research. Evasive attack and label flipping attacks are the two types. To trick the detection model, we "flipped" the label of malware to make it appear innocuous. On both binary and multi-class classification, we applied the same attack. As we raised the rate of labels for flipping, we noticed an increase in the fooling rate. SVM, LR, and MLP were the targets of this attack. SVM appears to perform more effectively than the other two models. In order to conduct a evasive attack, we locate a benign sample that is close to the SVM hyperplane. Identify the characteristics that have the biggest impact on the benign sample. We modified these characteristics in the malware testing samples. We used SVM, LR, and MLP to do our binary classification on these corrupted testing samples. When compared to the other two models, MLP fared better. Despite the fact that the model's fooling rate has increased because to the increased number of corrupted samples.

Several experiments have been carried out to examine the impact of the two attacks. The two adversarial example attacks can significantly reduce the performance of malware detection systems, experimental results overwhelmingly demonstrate. SVM has fared better in comparison to the other two models in label flipping attack and MLP has performed better in evasive attack. Therefore, SVM and MLP prevailed in our experiments.

Finally, we discuss a few potential extensions of the findings. We are looking into ways to defend against adversarial assaults, especially the data poisoning and evasion

attack, and to make malware detection systems stronger. Work with other adversarial attacks as well, such as algorithm logic, where the malware detection model's logic is distorted, and reverse engineering, where an adversary tries to recreate the classifier's decision limits.

Working on a reverse engineering attack is difficult, in our opinion, because the author will have restricted access to knowledge about the detection system and its data. On multiple-class classification, we wish to carry out the evasive attack. Due to the fact that MLP surpasses other machine learning models in terms of evasive attack, we also wish to experiment with additional deep learning methods.

LIST OF REFERENCES

[1] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, ''Effectiveness of opcode ngrams for detection of multi family android malware,'' in *2015 10th International Conference on Availability, Reliability and Security*, 2015, pp. 333--340.

[2] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, ''Crowdroid: Behavior-based malware detection system for android,'' in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. New York, NY, USA: Association for Computing Machinery, 2011, p. 15–26. [Online]. Available: https://doi-org.libaccess.sjlibrary.org/10.1145/2046614.2046619

[3] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, ''Drebin: Effective and explainable detection of android malware in your pocket,'' in *Network and Distributed System Security Symposium*, 02 2014.

[4] R. Samani, ''Mcafee mobile treat report,'' 2020. [Online]. Available: https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf

[5] B. Neha, A. Aemun, L. Wenjia, T. Fernanda, B. Arpit, and B. Prachi, ''Droidenemy: Battling adversarial example attacks for android malware detection,'' *Digital Communications and Networks*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864821000900

[6] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, ''Manipulating machine learning: Poisoning attacks and countermeasures for regression learning,'' in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 19--35.

[7] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, ''A comparison of static, dynamic, and hybrid analysis for malware detection,'' *Journal of Computer Virology and Hacking Techniques*, vol. 13, pp. 1--12, 2017.

[8] T. S. John and T. Thomas, ''Evading static and dynamic android malware detection mechanisms,'' in *Security in Computing and Communications*, S. M. Thampi, G. Wang, D. B. Rawat, R. Ko, and C.-I. Fan, Eds. Singapore: Springer Singapore, 2021, pp. 33--48.

[9] W. Schroeder, ''Learning machine learning part 2: Attacking white box models,'' 2022. [Online]. Available: https://posts.specterops.io/learning-machine-learning-part-2-attacking-white-box-models-1a10bbb4a2ae

[10] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, ‘‘Madam: Effective and efficient behavior-based android malware detection and prevention,’’ *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83--97, 2018.

[11] R. Hemant, S. Adithya, S. K. Sahay, and S. Mohit, ‘‘Robust malware detection models: Learning from adversarial attacks and defenses,’’ *Forensic Science International: Digital Investigation*, vol. 37, p. 301183, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721000913

[12] H. Rathore, S. Sahay, P. Nikam, and M. Sewak, ‘‘Robust android malware detection system against adversarial attacks using q-learning,’’ *Information Systems Frontiers*, vol. 23, pp. 1--16, 08 2021.

[13] C. Wang, L. Zhang, K. Zhao, X. Ding, and X. Wang, ‘‘Advandmal: Adversarial training for android malware detection and family classification,’’ *Symmetry*, vol. 13, no. 6, 2021. [Online]. Available: https://www.mdpi.com/2073-8994/13/6/1081

[14] A. Paudice, L. Muñoz-González, and E. C. Lupu, ‘‘Label sanitization against label flipping poisoning attacks,’’ in *ECML PKDD 2018 Workshops*, C. Alzate, A. Monreale, H. Assem, A. Bifet, T. S. Buda, B. Caglayan, B. Drury, E. García-Martín, R. Gavaldà, I. Koprinska, S. Kramer, N. Lavesson, M. Madden, I. Molloy, M.-I. Nicolae, and M. Sinn, Eds. Cham: Springer International Publishing, 2019, pp. 5--15.

[15] Y. Fahri Anıl and B. Şerif, ‘‘Data poisoning attacks against machine learning algorithms,’’ *Expert Systems with Applications*, vol. 208, p. 118101, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422012933

[16] B. Biggio, B. Nelson, and P. Laskov, ‘‘Poisoning attacks against support vector machines,’’ in *Proceedings of the 29th International Conference on Machine Learning*, ser. ICML’12. Madison, WI, USA: Omnipress, 2012, p. 1467–1474.

[17] H. Bostani and V. Moonsamy, ‘‘Evadedroid: A practical evasion attack on machine learning for black-box android malware detection,’’ 2021. [Online]. Available: https://arxiv.org/abs/2110.03301

[18] M. Stamp, *Introduction to Machine Learning with Applications in Information Security*, 2nd ed. Chapman & Hall, 2022.

[19] Swarnimrai, ‘‘Multi-layer perceptron learning in tensorflow,’’ 2021. [Online]. Available: https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/

[20] S. Mark, ‘‘Alphabet soup of deep learning topics,’’ 2019. [Online]. Available: https://www.cs.sjsu.edu/~stamp/RUA/alpha.pdf

[21] K. Xu, Y. Li, R. H. Deng, and K. Chen, ‘‘Deeprefiner: Multi-layer android malware detection system applying deep neural networks,’’ in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018, pp. 473--487.

[22] A. Agrawal, ‘‘Logistic regression. simplified.’’ 2017. [Online]. Available: https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389

[23] M. Grandini, E. Bagli, and G. Visani, ‘‘Metrics for multi-class classification: an overview,’’ 2020. [Online]. Available: https://arxiv.org/abs/2008.05756

[24] G.-M. Alejandro, B. Hayretdin, and N. Sven, ‘‘Kronodroid: Time-based hybrid-featured dataset for effective android malware detection and characterization,’’ *Computers & Security*, vol. 110, p. 102399, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002236

[25] M. Spreitzenbarth and J. Bombien, ‘‘Forensic blog - mobile phone forensics and mobile malware,’’ 2015. [Online]. Available: https://forensics.spreitzenbarth.de/android-malware/

[26] ‘‘Riskware:android/smsreg.’’ [Online]. Available: https://www.f-secure.com/sw-desc/riskware_android_smsreg.shtml

[27] ‘‘Android/trojan.agent.’’ [Online]. Available: https://www.malwarebytes.com/blog/detections/android-trojan-agent/

[28] P. Danny, ‘‘Bankbot android malware sneaks into the google play store - for the third time,’’ 2017. [Online]. Available: https://www.zdnet.com/article/bankbot-android-malware-sneaks-into-the-google-play-store-for-the-third-time/

APPENDIX A

Label Flip Attack on Binary Classification

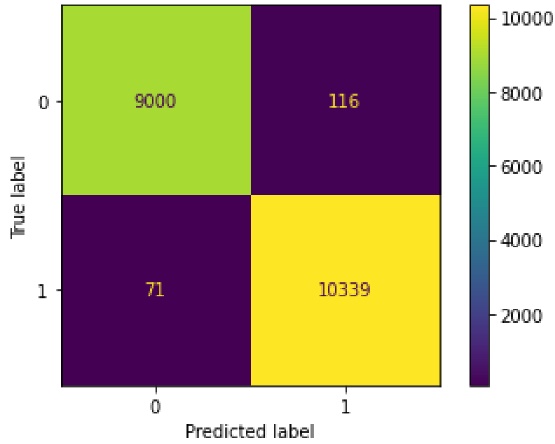

Figure A.23: SVM Confusion Matrix before Label Flip Attack

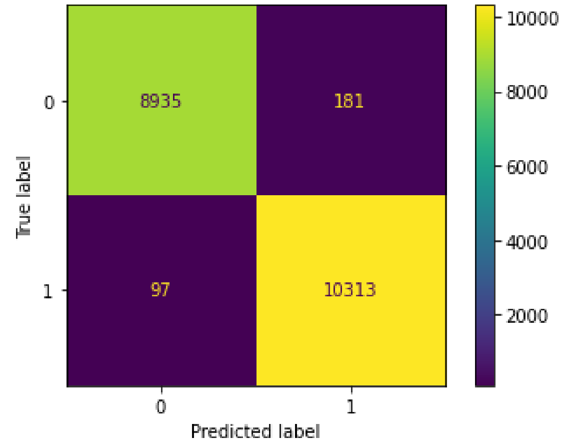

Figure A.24: SVM Confusion Matrix with 10% Label Flip Attack

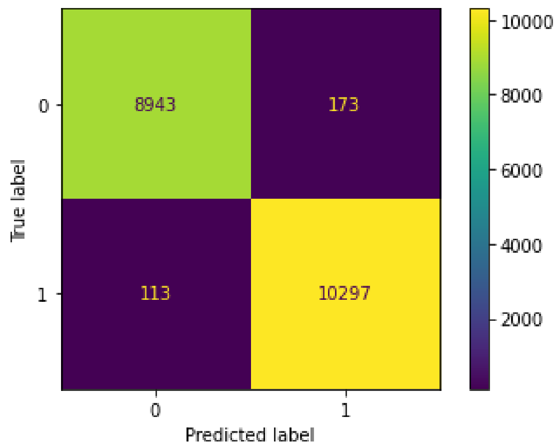

Figure A.25: SVM Confusion Matrix with 20% Label Flip Attack

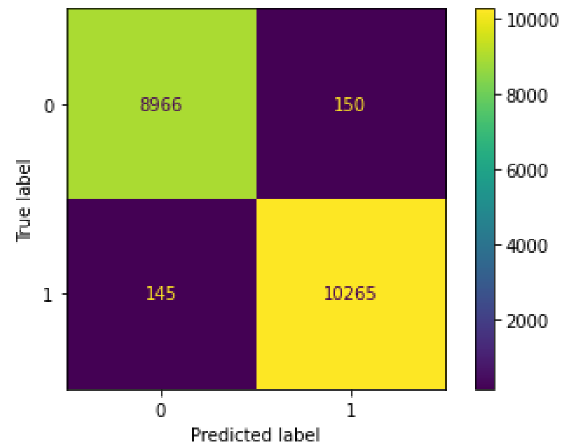

Figure A.26: SVM Confusion Matrix with 30% Label Flip Attack

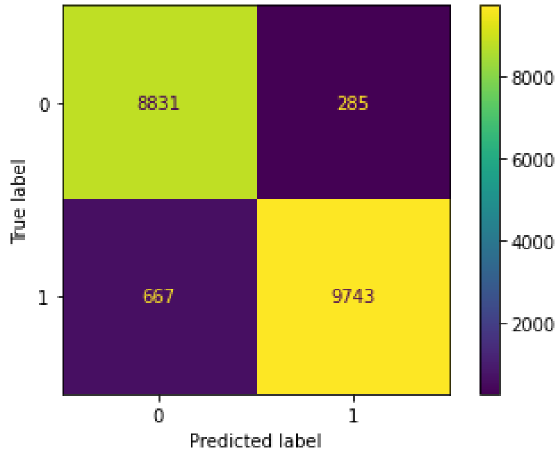

Figure A.27: SVM Confusion Matrix with 40% Label Flip Attack

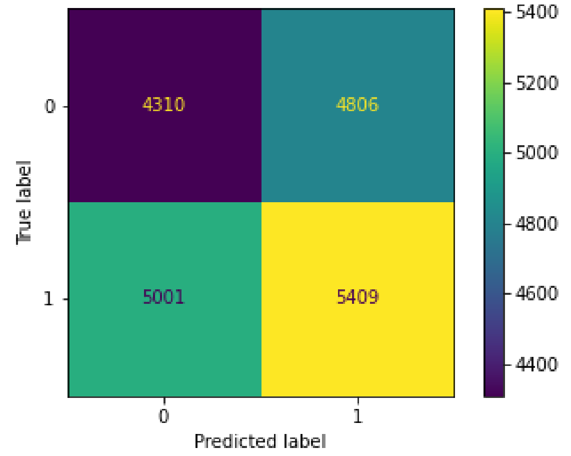

Figure A.28: SVM Confusion Matrix with 50% Label Flip Attack

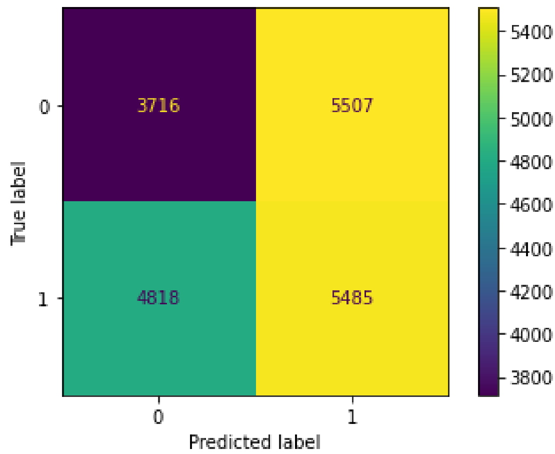

Figure A.29: SVM Confusion Matrix with 60% Label Flip Attack

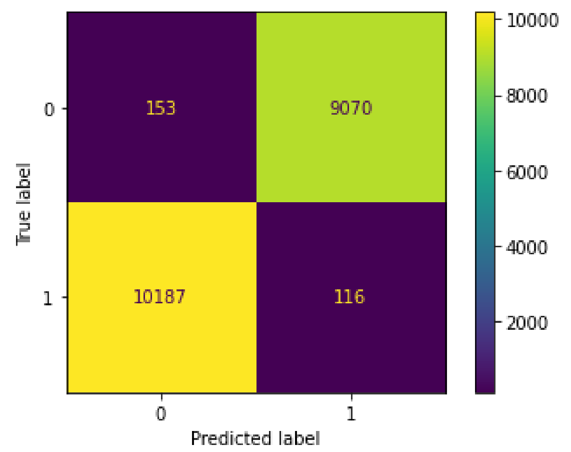

Figure A.30: SVM Confusion Matrix with 70% Label Flip Attack

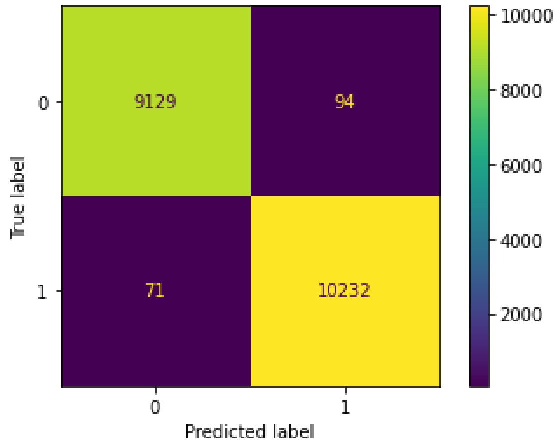

Figure A.31: LR Confusion Matrix before Label Flip Attack

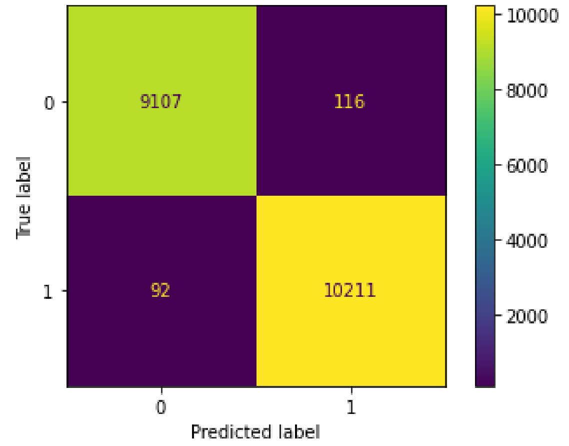

Figure A.32: LR Confusion Matrix with 10% Label Flip Attack

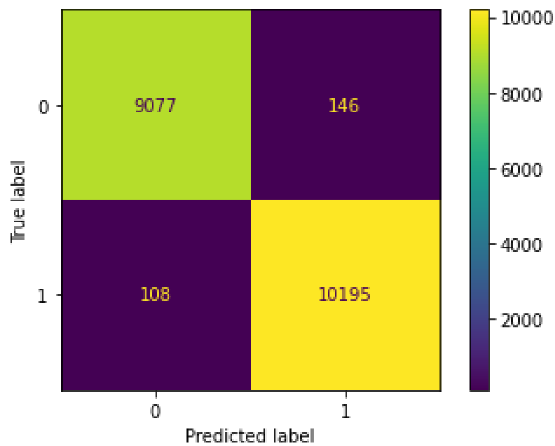

Figure A.33: LR Confusion Matrix with 20% Label Flip Attack

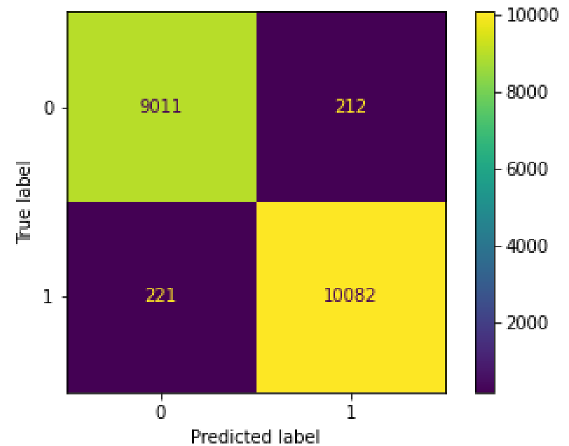

Figure A.34: LR Confusion Matrix with 30% Label Flip Attack

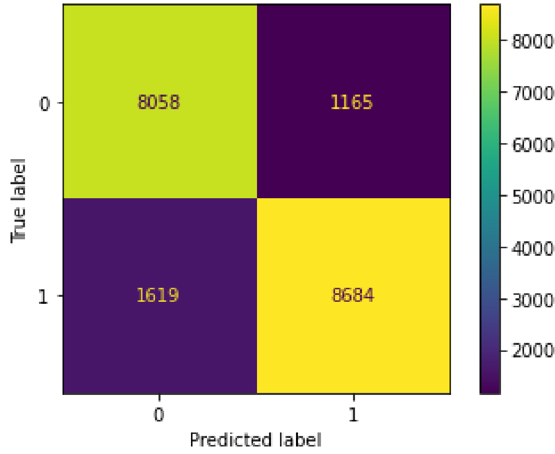

Figure A.35: LR Confusion Matrix with 40% Label Flip Attack

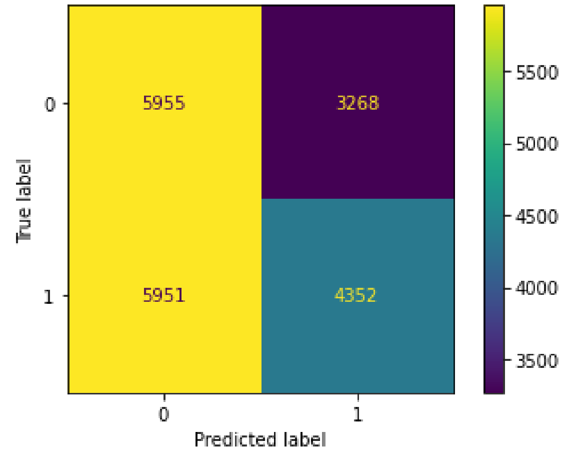

Figure A.36: LR Confusion Matrix with 50% Label Flip Attack

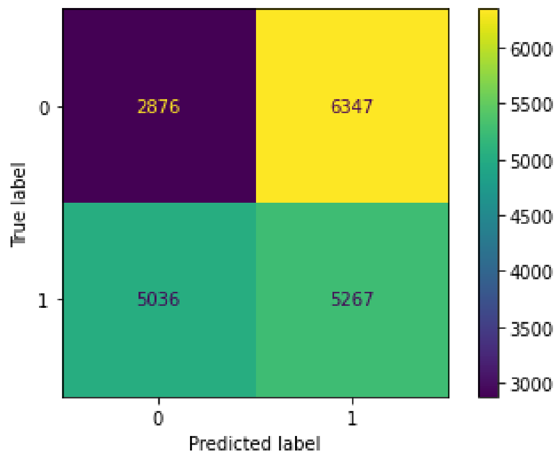

Figure A.37: LR Confusion Matrix with 60% Label Flip Attack

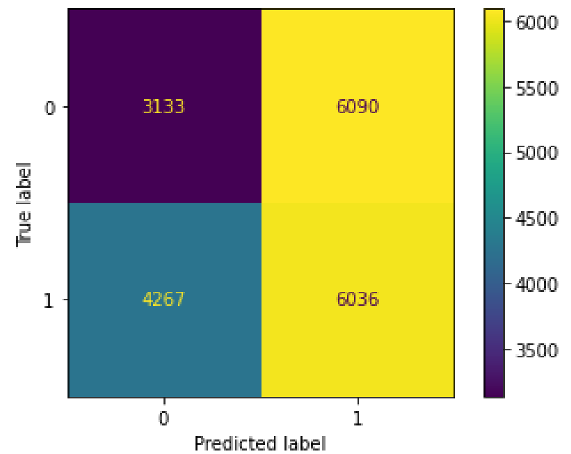

Figure A.38: LR Confusion Matrix with 70% Label Flip Attack

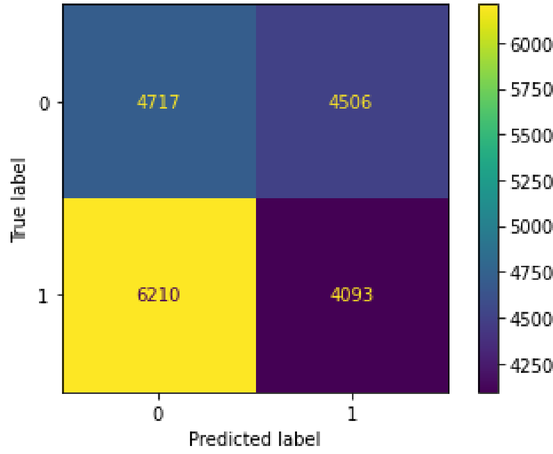

Figure A.39: LR Confusion Matrix with 80% Label Flip Attack

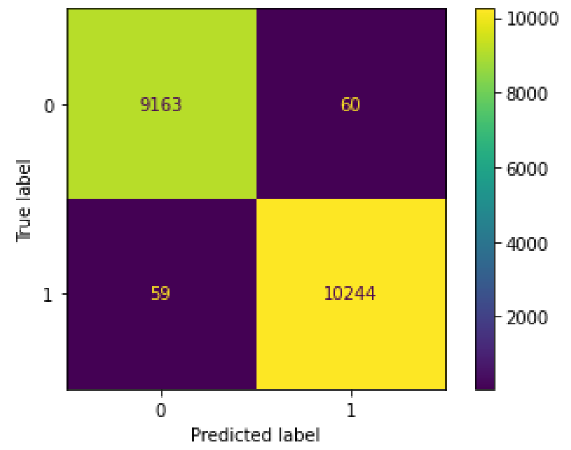

Figure A.40: MLP Confusion Matrix before Label Flip Attack

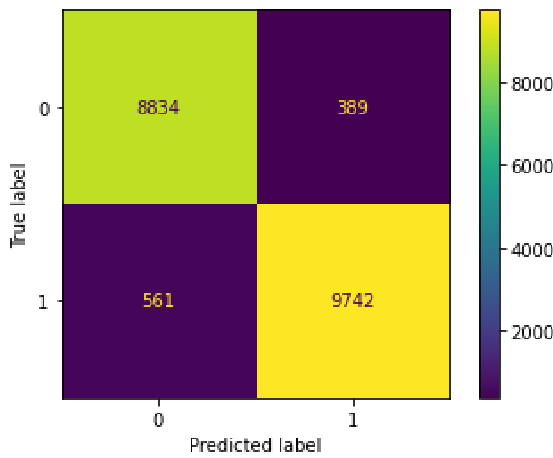

Figure A.41: MLP Confusion Matrix with 10% Label Flip Attack

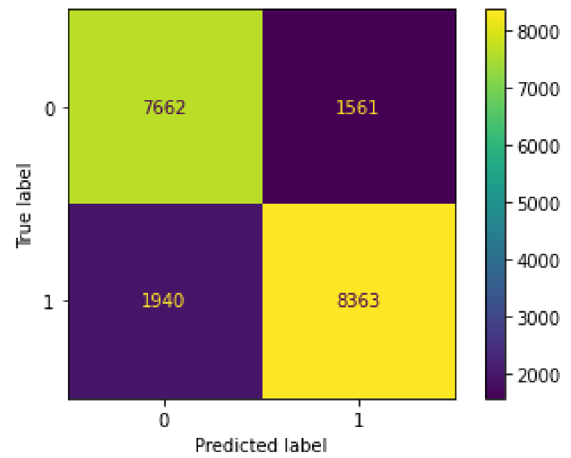

Figure A.42: MLP Confusion Matrix with 20% Label Flip Attack

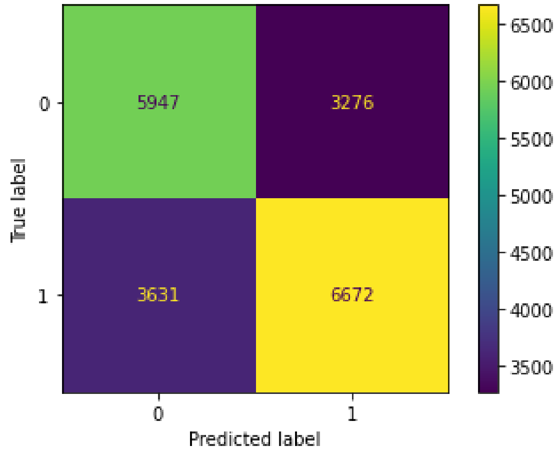

Figure A.43: MLP Confusion Matrix with 30% Label Flip Attack

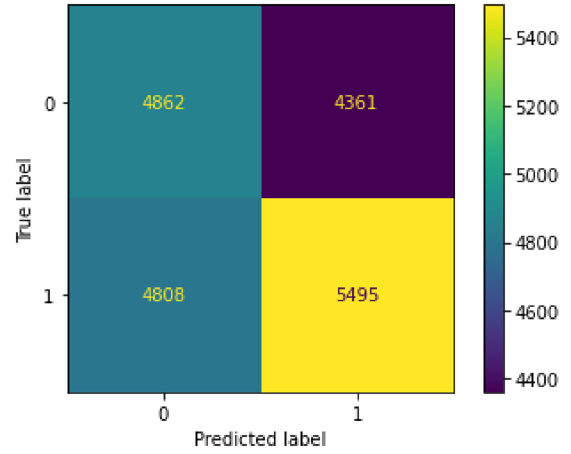

Figure A.44: MLP Confusion Matrix with 40% Label Flip Attack

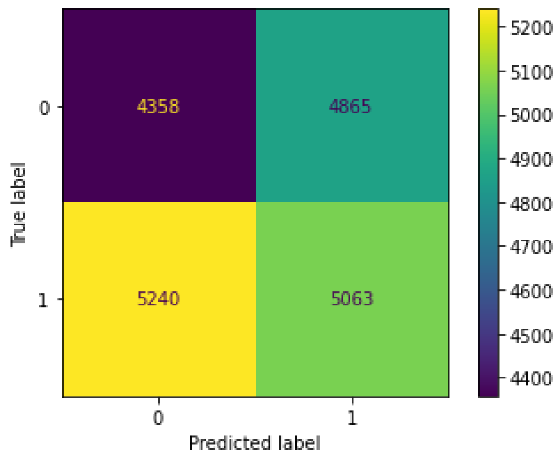

Figure A.45: MLP Confusion Matrix with 50% Label Flip Attack

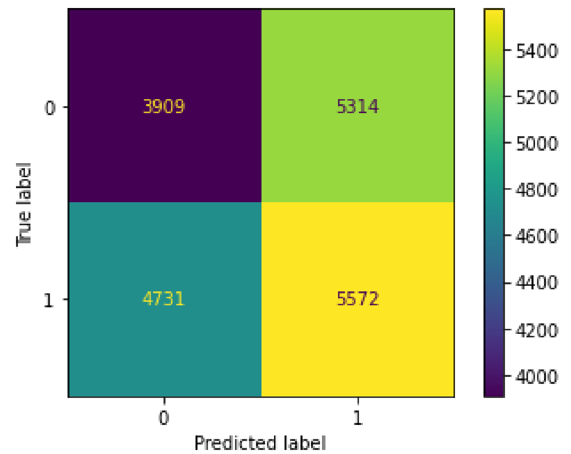

Figure A.46: MLP Confusion Matrix with 60% Label Flip Attack

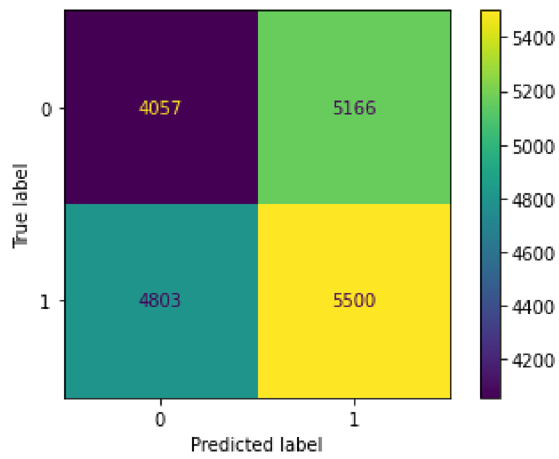

Figure A.47: MLP Confusion Matrix with 70% Label Flip Attack

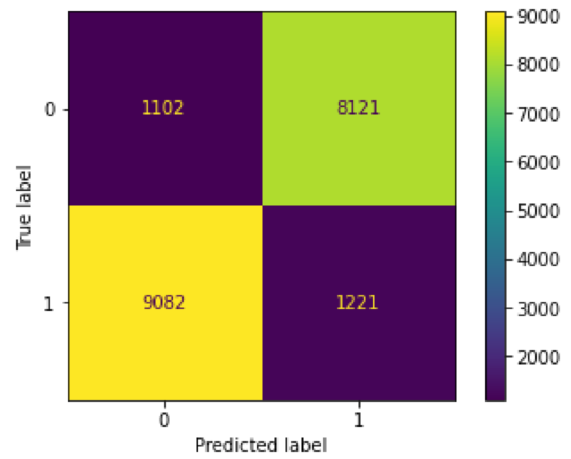

Figure A.48: MLP Confusion Matrix with 80% Label Flip Attack

APPENDIX B

Label Flip Attack on Multi-Class Classification

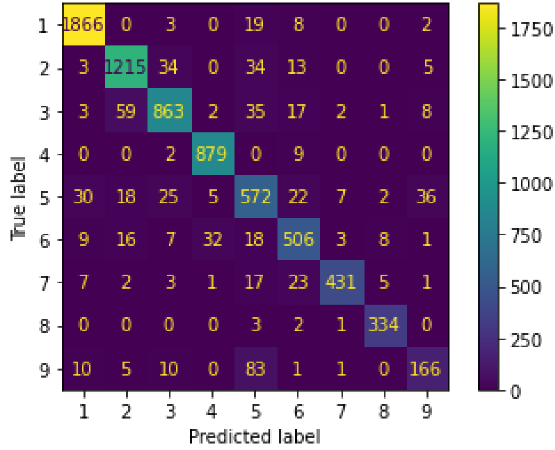

Figure B.49: SVM Confusion Matrix before Label Flip Attack

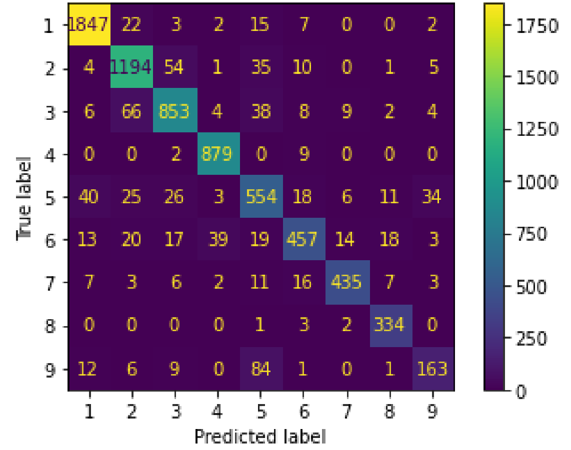

Figure B.50: SVM Confusion Matrix with 10% Label Flip Attack

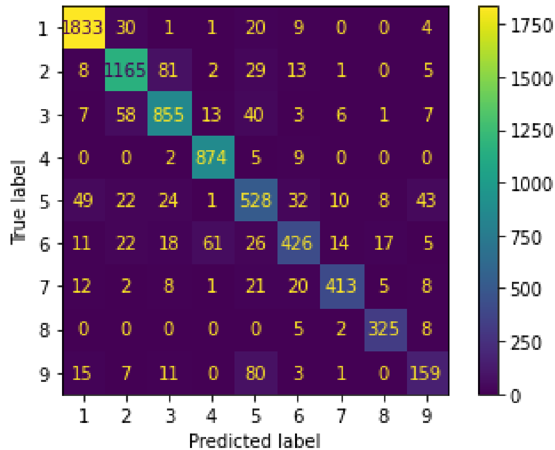

Figure B.51: SVM Confusion Matrix with 20% Label Flip Attack

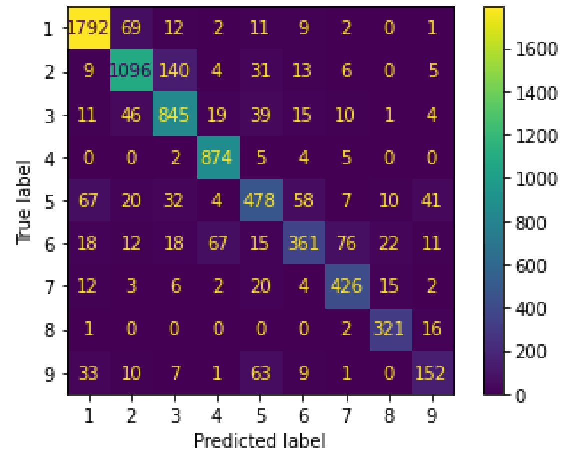

Figure B.52: SVM Confusion Matrix with 30% Label Flip Attack

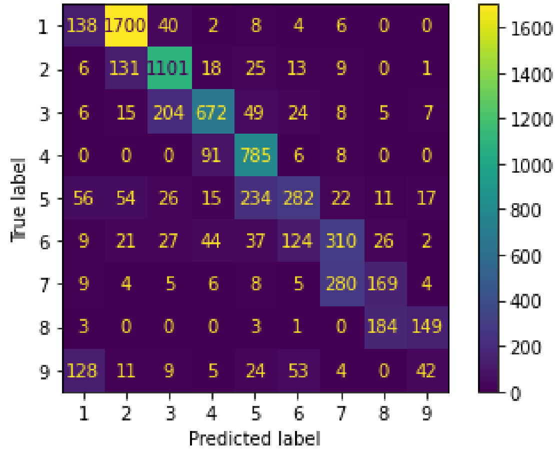

Figure B.53: SVM Confusion Matrix with 40% Label Flip Attack

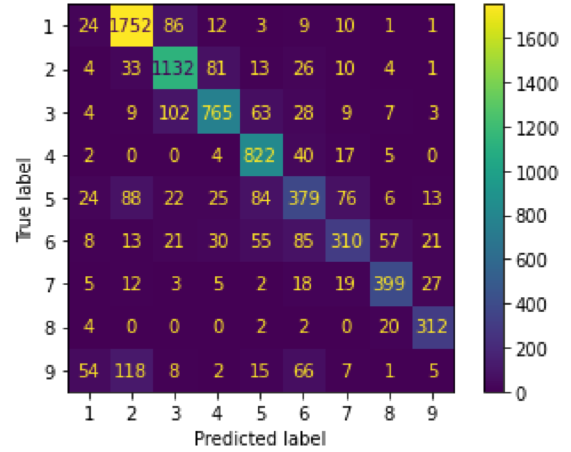

Figure B.54: SVM Confusion Matrix with 50% Label Flip Attack

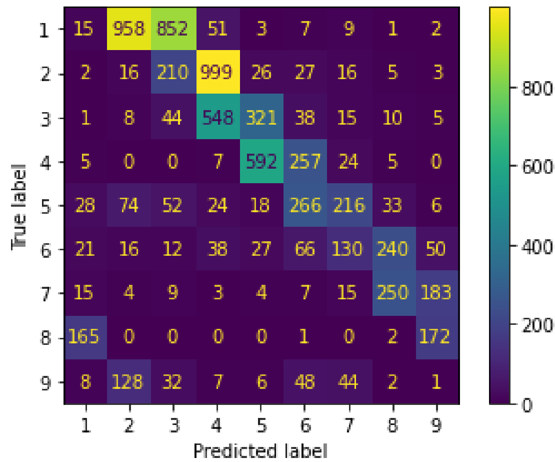

Figure B.55: SVM Confusion Matrix with 60% Label Flip Attack

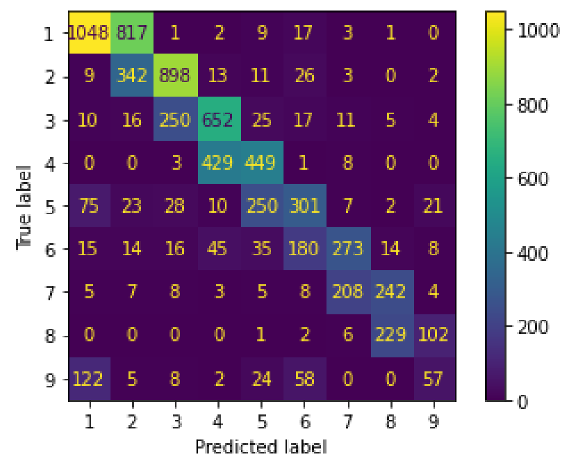

Figure B.56: SVM Confusion Matrix with 70% Label Flip Attack

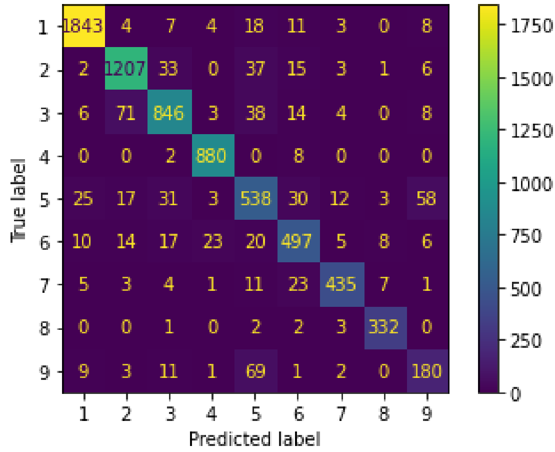

Figure B.57: LR Confusion Matrix before Label Flip Attack

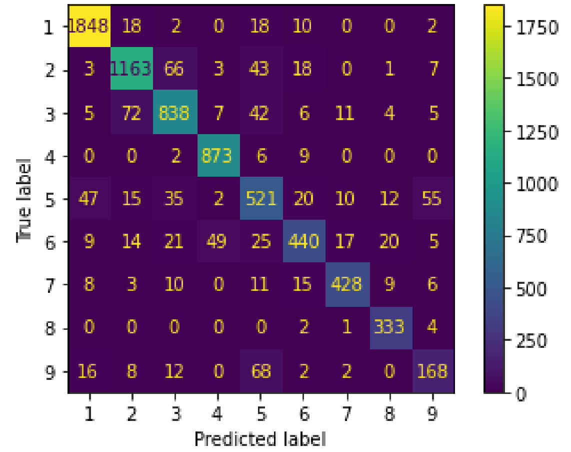

Figure B.58: LR Confusion Matrix with 10% Label Flip Attack

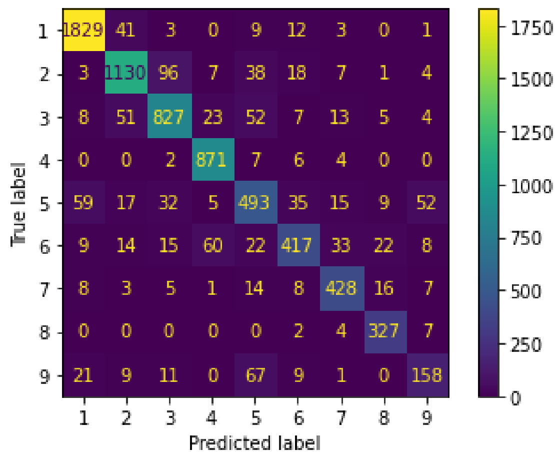

Figure B.59: LR Confusion Matrix with 20% Label Flip Attack

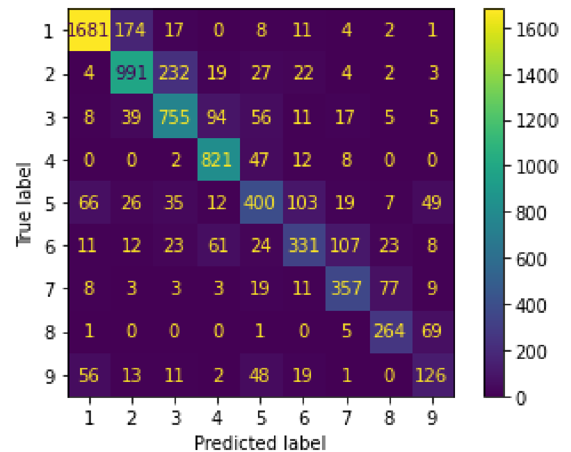

Figure B.60: LR Confusion Matrix with 30% Label Flip Attack

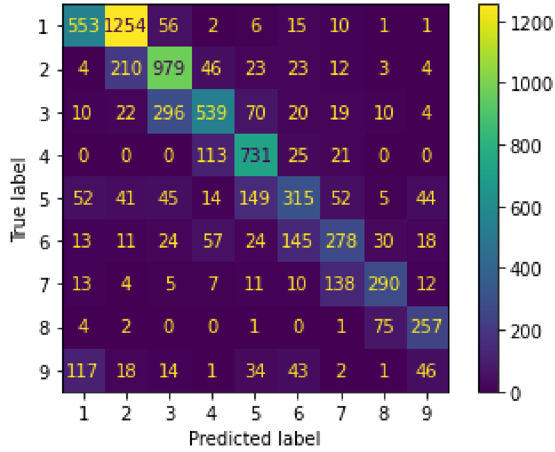

Figure B.61: LR Confusion Matrix with 40% Label Flip Attack

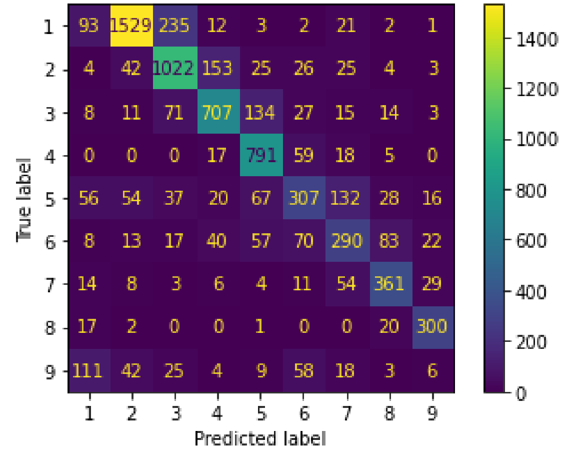

Figure B.62: LR Confusion Matrix with 50% Label Flip Attack

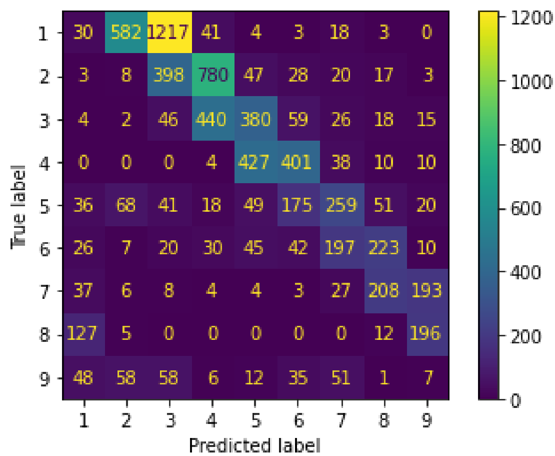

Figure B.63: LR Confusion Matrix with 60% Label Flip Attack

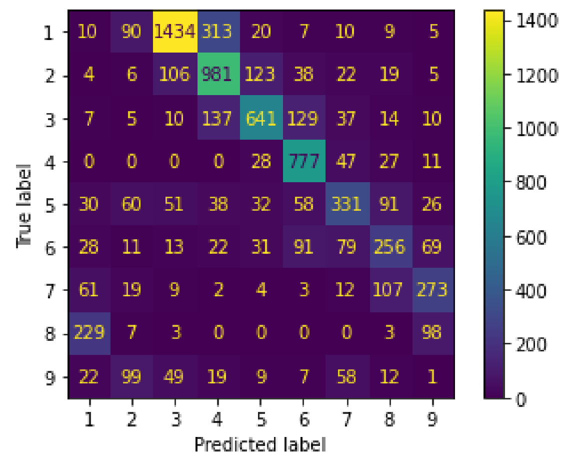

Figure B.64: LR Confusion Matrix with 70% Label Flip Attack

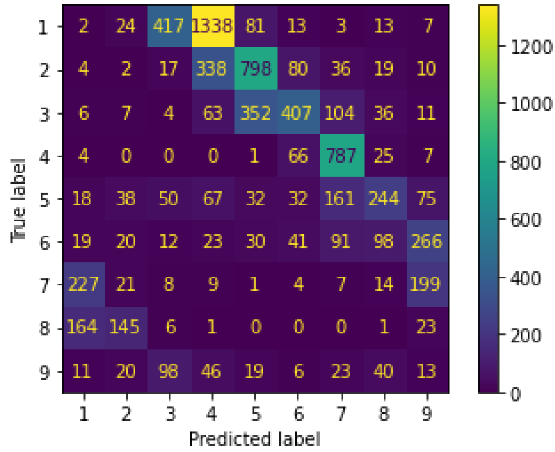

Figure B.65: LR Confusion Matrix with 80% Label Flip Attack

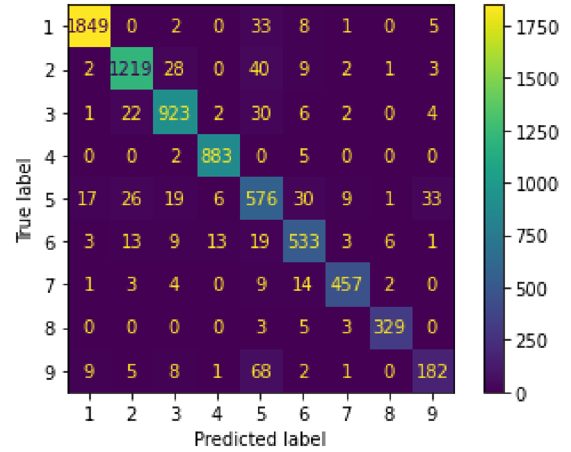

Figure B.66: MLP Confusion Matrix before Label Flip Attack

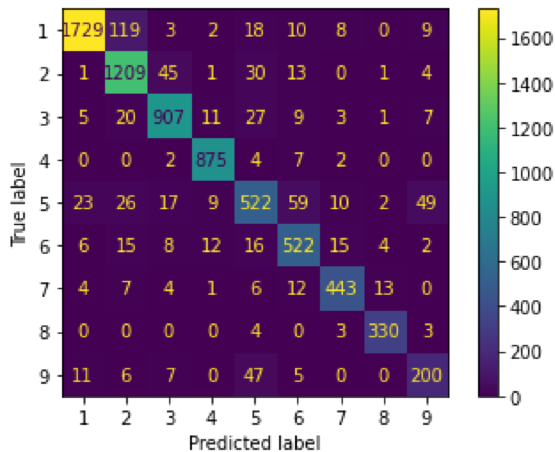

Figure B.67: MLP Confusion Matrix with 10% Label Flip Attack

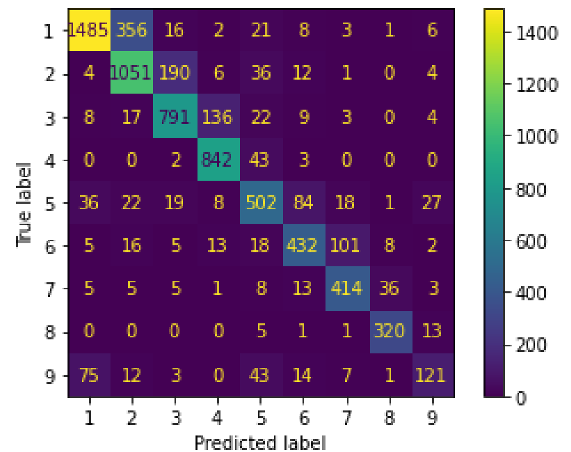

Figure B.68: MLP Confusion Matrix with 20% Label Flip Attack

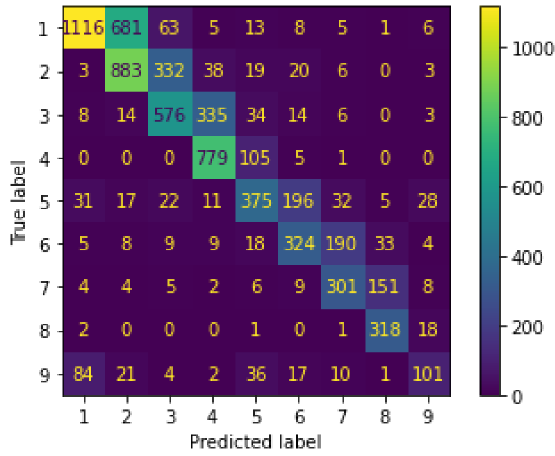

Figure B.69: MLP Confusion Matrix with 30% Label Flip Attack

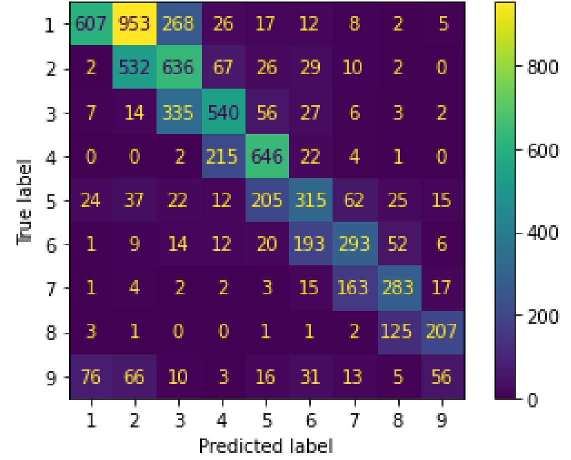

Figure B.70: MLP Confusion Matrix with 40% Label Flip Attack

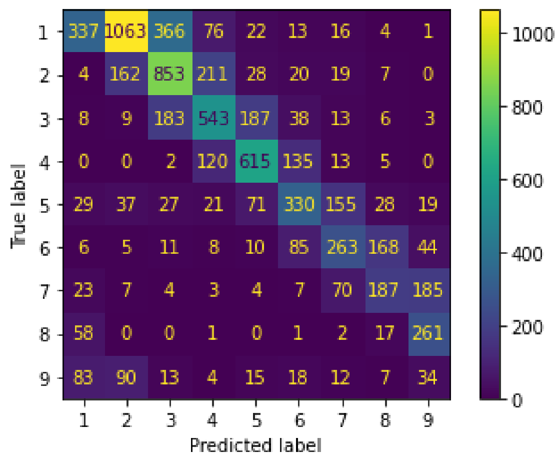

Figure B.71: MLP Confusion Matrix with 50% Label Flip Attack

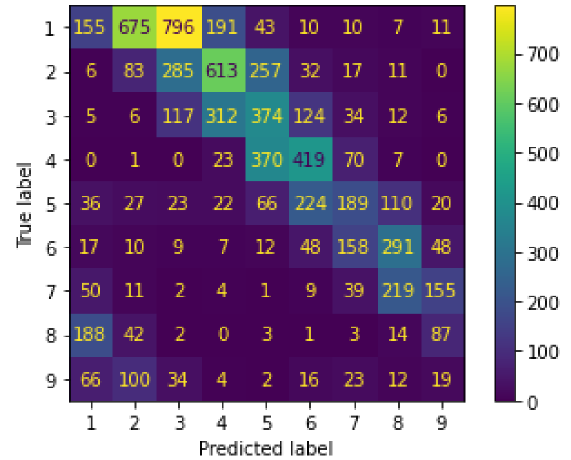

Figure B.72: MLP Confusion Matrix with 60% Label Flip Attack

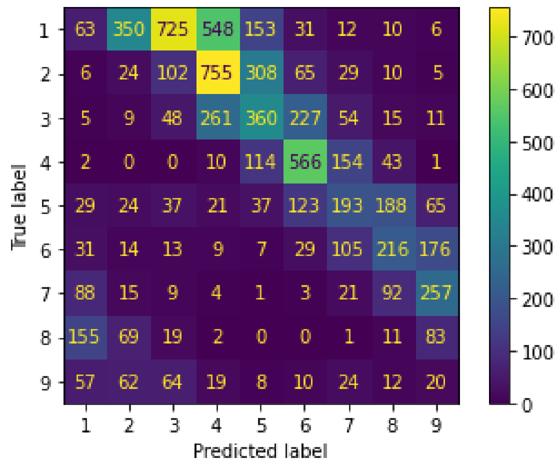

Figure B.73: MLP Confusion Matrix with 70% Label Flip Attack

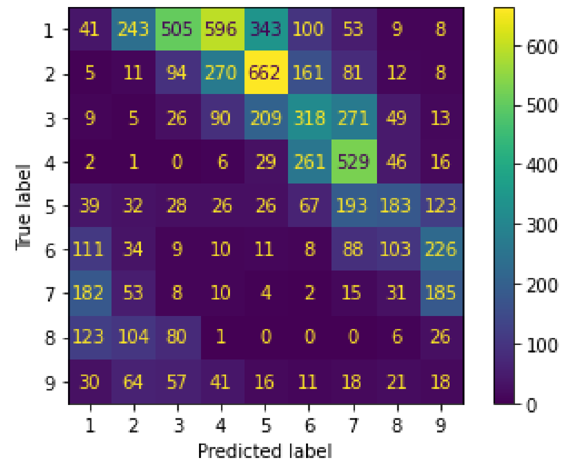

Figure B.74: MLP Confusion Matrix with 80% Label Flip Attack

APPENDIX C

Evasive Attack on Binary Classification

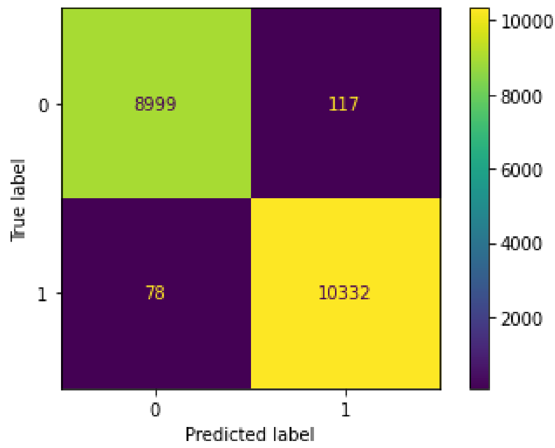

Figure C.75: SVM Confusion Matrix before Evasive Attack

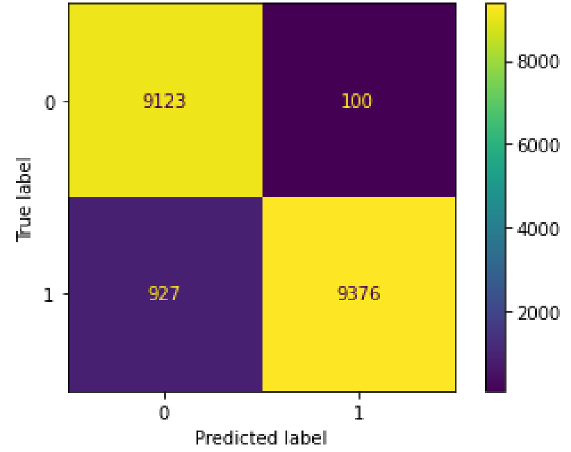

Figure C.76: SVM Confusion Matrix with 10% Evasive Attack

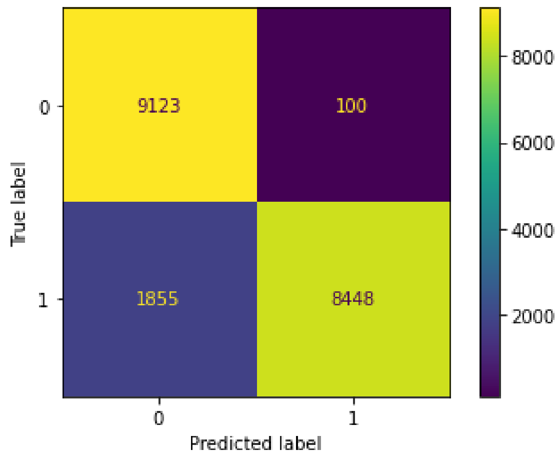

Figure C.77: SVM Confusion Matrix with 20% Evasive Attack

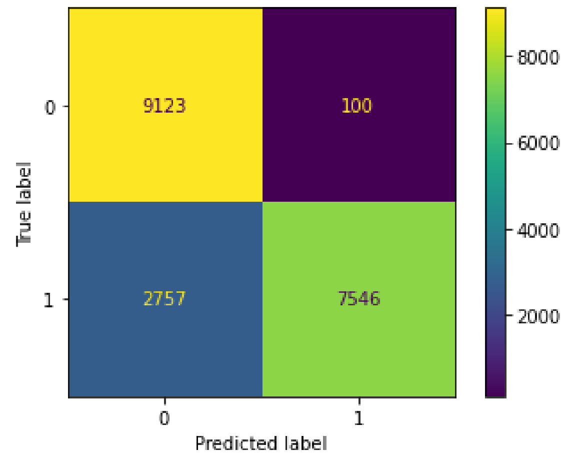

Figure C.78: SVM Confusion Matrix with 30% Evasive Attack

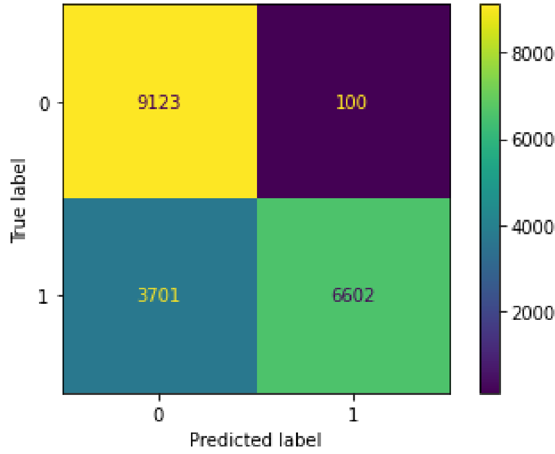

Figure C.79: SVM Confusion Matrix with 40% Evasive Attack

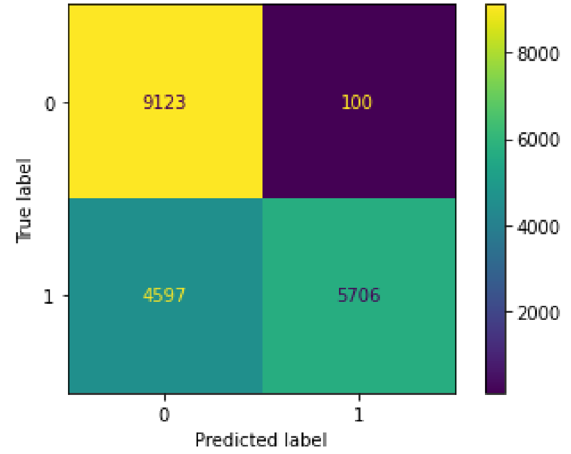

Figure C.80: SVM Confusion Matrix with 50% Evasive Attack

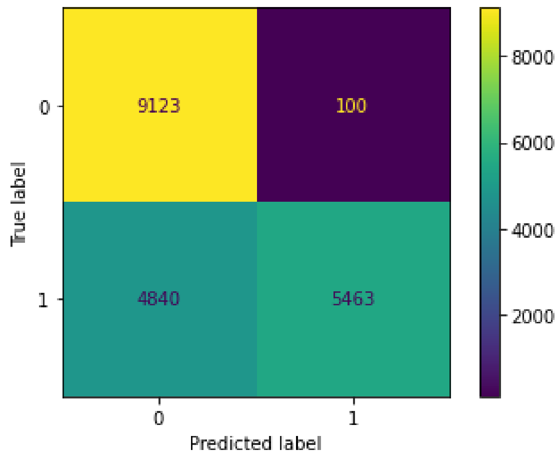

Figure C.81: SVM Confusion Matrix with 60% Evasive Attack

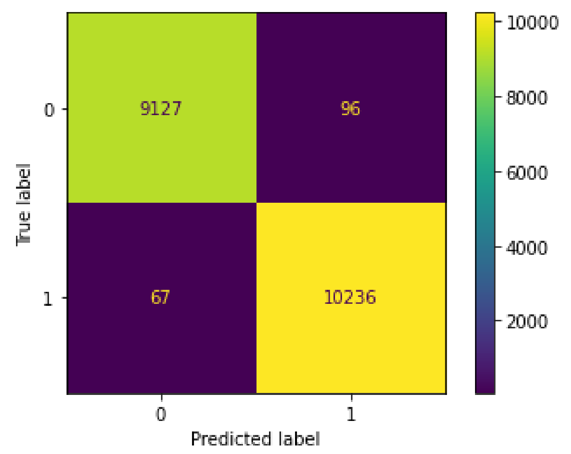

Figure C.82: LR Confusion Matrix before Evasive Attack

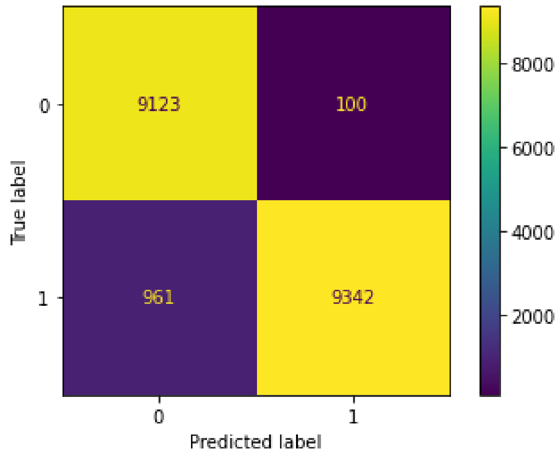

Figure C.83: LR Confusion Matrix with 10% Evasive Attack

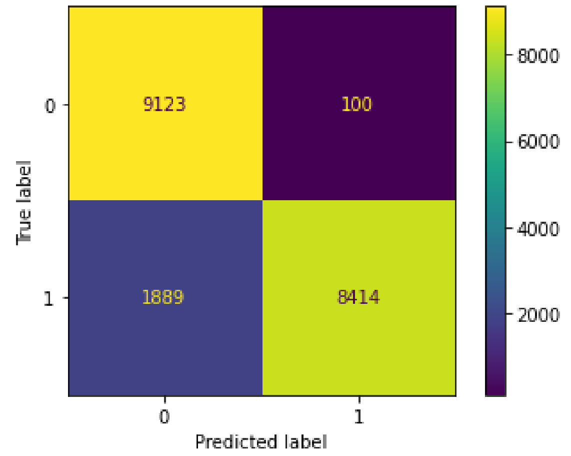

Figure C.84: LR Confusion Matrix with 20% Evasive Attack

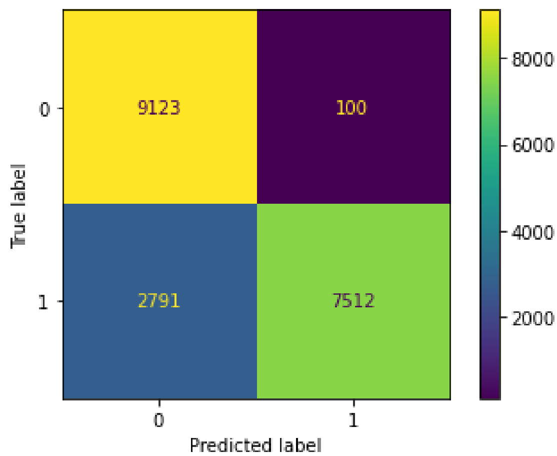

Figure C.85: LR Confusion Matrix with 30% Evasive Attack

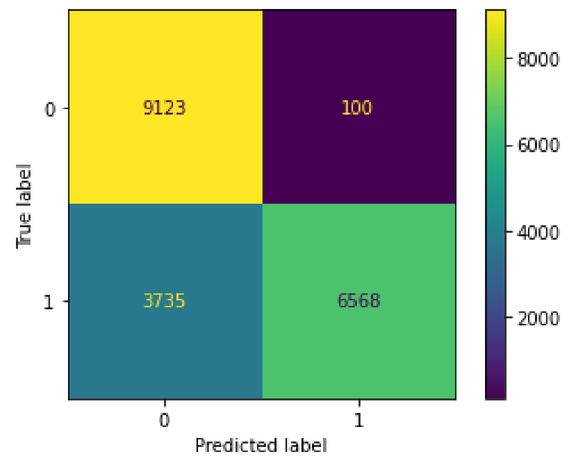

Figure C.86: LR Confusion Matrix with 40% Evasive Attack

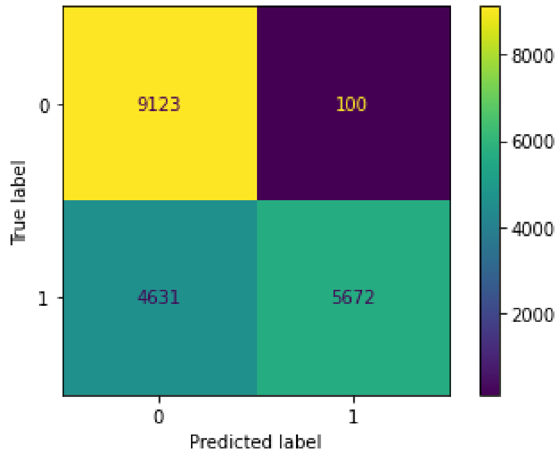

Figure C.87: LR Confusion Matrix with 50% Evasive Attack

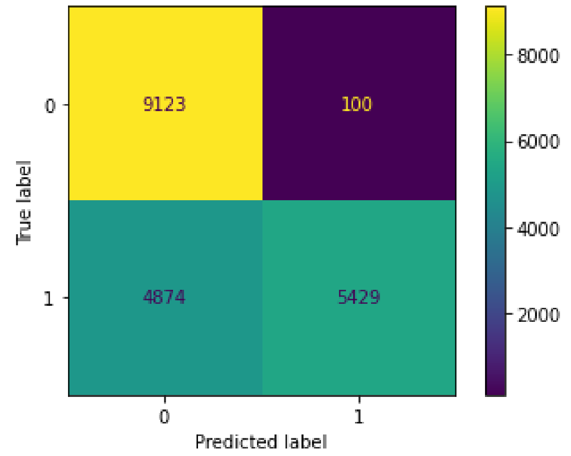

Figure C.88: LR Confusion Matrix with 60% Evasive Attack

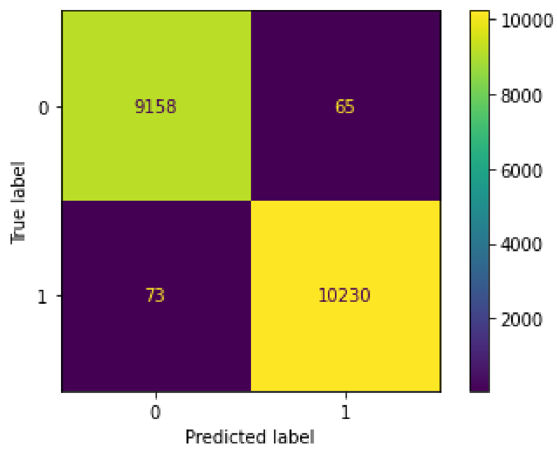

Figure C.89: MLP Confusion Matrix before Evasive Attack

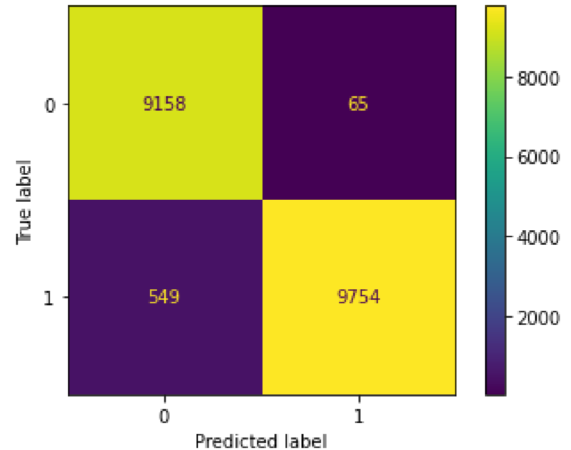

Figure C.90: MLP Confusion Matrix with 10% Evasive Attack

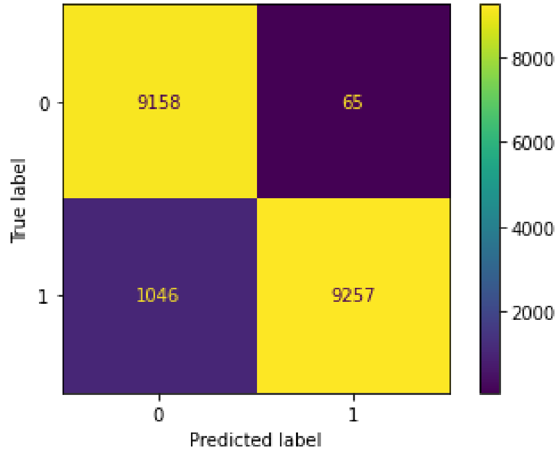

Figure C.91: MLP Confusion Matrix with 20% Evasive Attack

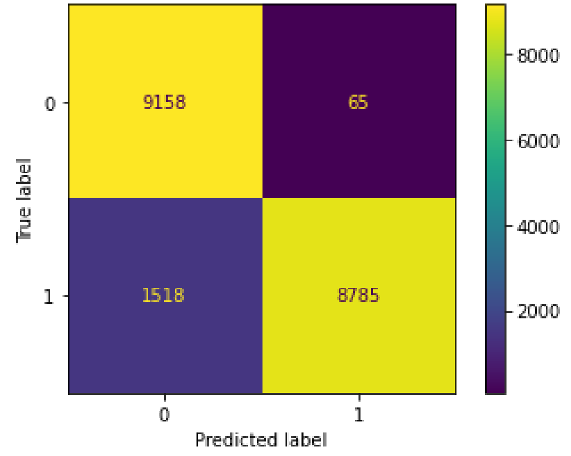

Figure C.92: MLP Confusion Matrix with 30% Evasive Attack

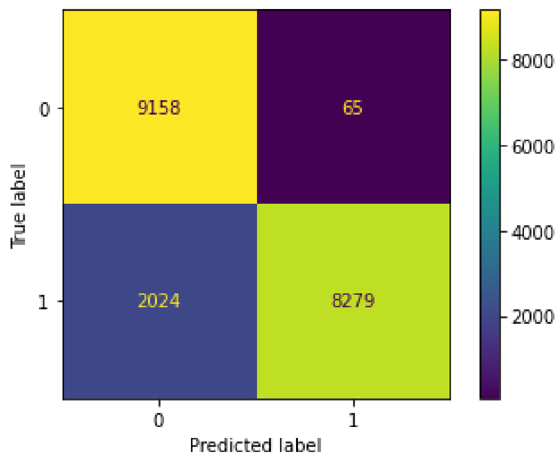

Figure C.93: MLP Confusion Matrix with 40% Evasive Attack

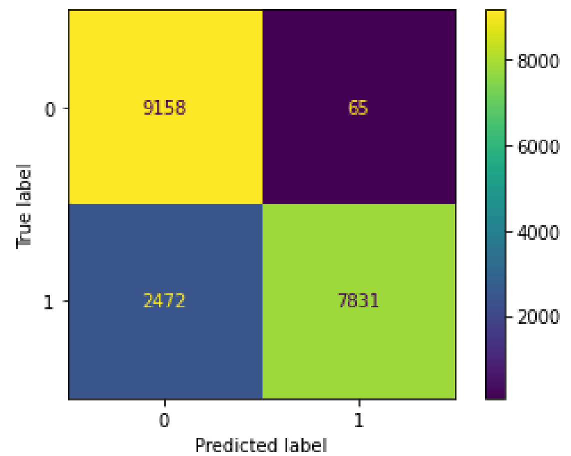

Figure C.94: MLP Confusion Matrix with 50% Evasive Attack