Research Project Report

on

Developing robust machine learning model to detect adversarial attacks

Submitted in Partial fulfillment for the award of honor degree in Bachelor in Computer Science

Submitted by

Anushka Rana (20CS-01)

Shatrughan Gusain (20CS-20)

Shashank Gusain (20CS-14)

Under the Supervision of

Dr. Preeti Mishra

Assistant Professor

Department of Computer Science

Doon University

Dehradun, Uttrakhand

June, 2023

DECLARATION

We hereby declare that the project report entitled "**Developing robust machine learning model to detect adversarial attacks**" being submitted to the Department of Computer Science, School of Physical Science, Doon University, Dehradun in partial fulfillment of the requirements for the award of the degree of **Bachelor of Science Hons.(Computer Science)** contains the original work carried out by us under the supervision of **Dr. Preeti Mishra** .

Shashank Gusain (20CS-14)

Anushka Rana (20CS-01)

Shatrughan Singh Gusain (20CS-20)

ACKNOWLEDGEMENT

We express our deep gratitude to our project supervisor Dr. Preeti Mishra for giving us this wonderful opportunity to work under her on the project "**Developing robust machine learning model to detect adversarial attacks**", who always encouraged us to think independently and to develop a "problem-solving" attitude. Her constant support on understanding the underlying theory behind our experimental results enabled us to develop an appreciation of the subject.

We would like to express our deepest gratitude to our lab mates and our friends for their undying faith in our capabilities and for their continuous support in realizing our academic goals. Finally, we offer our regards and thanks to all others who supported us in any aspect during our project and whose names we might have missed here.

Shashank Gusain (20CS-14)

Anushka Rana (20CS-01)

Shatrughan Singh Gusain (20CS-20)

CERTIFICATE

This is to certify that the research project entitled **Developing robust machine learning model to detect adversarial attacks** submitted by **Shashank Gusain (20CS-14), Anushka Rana (20CS-01), Shatrughan Singh Gusain (20CS-20)** to the Department of Computer Science, School of Physical Science, Doon University, Dehradun in partial fulfillment of the requirements for the report of the project, is a record of the bonafide research work carried out by them under my supervision and guidance.

The results embodied in the research project have not been submitted to any other university or institute for the award of any degree or diploma.

Dr. Narendra Rawal
(Head of the Department)
Department of Computer Science
School of Physical Science
Doon University, Dehradun, India

Dr. Preeti Mishra
(Supervisor)
Department of Computer Science
School of Physical Science
Doon University, Dehradun, India

ABSTRACT

The primary goal of virtualisation is to extract the fundamental physical infrastructure and develop virtual environments that are isolated and independent from one another. This isolation provides enhanced security as issues with any one of the virtual instance would not affect the others. Research works lately have uncovered numerous expected security concerns and vulnerabilities in virtual environments. Addressing these vulnerabilities needs planning, security measures and monitoring of these virtual environments.

Windows provides various virtualization solutions that enable users to create and run virtual machines (VMs) on their Windows-based systems, allowing users to run multiple operating systems and applications simultaneously on a single physical computer. While Windows-based virtualization offers numerous benefits, it is important to be aware of potential security threats that can affect virtualized environments.

Adversarial machine learning (AML) can play a crucial role in securing virtual environments by identifying and mitigating potential threats and attacks. It involves exploring the manipulation and exploitation of these models through the introduction of specially crafted input data, known as adversarial examples or attacks. While AML can provide valuable insights and detection capabilities, adversarial attacks can be designed to evade AML defenses therefore it is not a foolproof solution.

The project research begins by analyzing various adversarial attack techniques and understanding their potential impact on machine learning models. Based on this analysis, a defense mechanism is proposed to enhance the

model's resilience against adversarial attacks. The defense mechanism involves adversarial training, where the model is trained using both clean and adversarial examples to improve its ability to detect adversarial inputs.

Keywords: Evasion attack, poisoning attack, adversarial robustness tools, adversarial machine learning models

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A report[1] published by Ani Petrosyan in may, 2023 suggested that the number of malware attacks all over the world reached 5.5 billion in the year 2022, where it was observed an increment of about 2 percent than the previous year. 1.3.

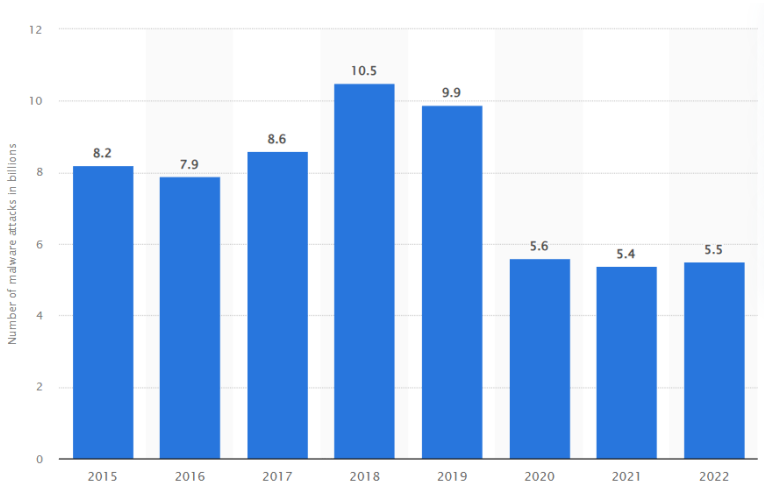

Figure 1.1: Number of malware attacks all over world from 2015-2022(in billions)

[1]https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/

The highest number of cases of malware attacks were recorded in the year 2018 with 10.5 billion reported attacks worldwide. Out of which, 58 percent of computer malware is known to be Trojan.[2]

According to statista, the most widely used tool used by the cybercriminals to infiltrate windows systems were Trojan.[3]. As measured, about 64.31 percent of all malware attacks on windows systems was caused using Trojan (figure 1.2).

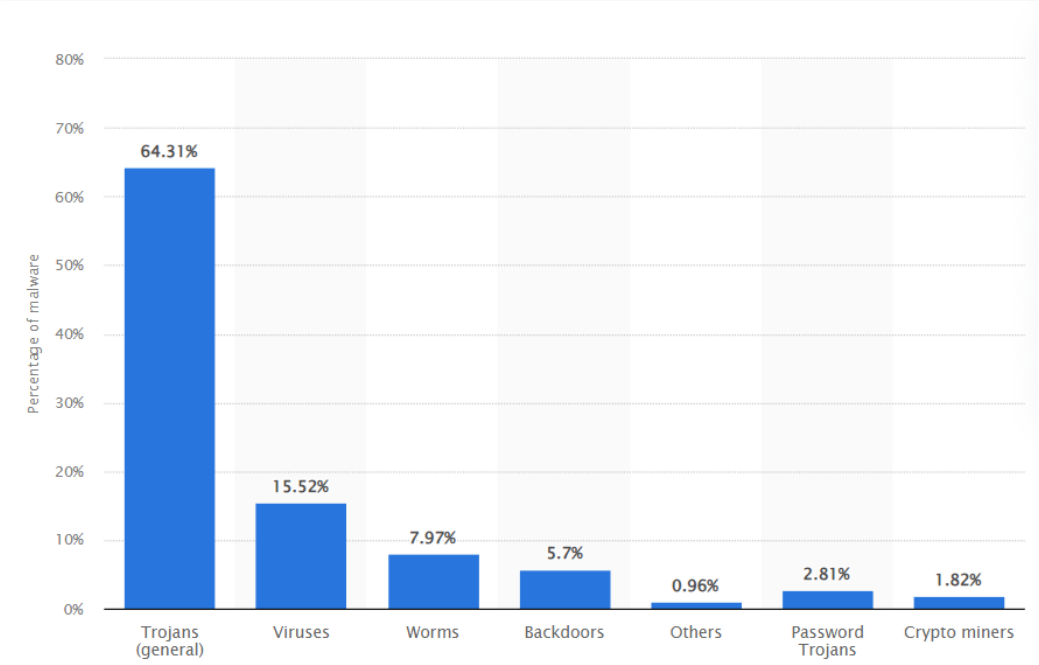

Figure 1.2: Distribution of leading malware types

In today's world, virtual security or cyber security is one of the most important aspect to keep our data secure online.

Virtual security is the process of preventing unauthorized access or attack on computer networks and data. It includes equipment and programming

[2]https://www.worthinsurance.com/post/malware-statistics
[3]https://www.statista.com/statistics/221506/share-of-new-types-of-malware/

transformations, strategies, and techniques planned to secure networks from unauthorised clients. Virtual security refers to the assurance of advanced networks, organizations, and data from unapproved access, harm, or robbery online.

Virtualization can assist with further developing security in more than one way.

- Initially, businesses can divide their networks using server virtualization to separate sensitive data from less secure parts of the network. This reduces the risks of information theft and makes it much easier to fix any issues.

- Secondly, by making secure network connection and management easier, betterment in security can also be achieved through network virtualization. It becomes simpler to monitor and control whatever happens on a computer's VPN when all network traffic passes through a centralized gateway.

- Finally, security enhancements can also be achieved through desktop virtualisation by making it simpler to manage and secure desktop systems. If all the data and applications are stored on a centralised server, businesses can easily make sure that only those who are allowed can access to it.

The key features of virtual security are:

Authentication: It is a method to confirm the nature of the client or the network in a virtual environment. Passwords, biometrics, two-factor authentication (2FA), or other multi-factor authentication methods are typically involved.

Encryption: The act of transforming data into a format that cannot be read in order to prevent unauthorized access. It secures sensitive data during capacity and transmission, for example, Secure Sockets Layer (SSL) and Transport Layer Security (TLS).

Firewalls: Networks and systems can be secured from unauthorized access using firewalls. They screen and control the approaching network traffic in view of already determined security rules.

Virtual private networks (VPNs): VPNs lay out secure networks over public organizations, like the web, to guarantee protection and safeguard data during its transmission. They are ordinarily used to make secure remote access for clients and to get interchanges between various areas.

Patch management: Consistently applying security fixes and updates to programming, working networks, and applications is required to address weaknesses and safeguard against known security threats.

Security Awareness and training: Teaching clients about security practices, for example, making strong passwords, distinguishing phishing attempts, and keeping away from doubtful downloads, has a fundamental part in virtual security.

Data Recovery and Backup: Important data is secured from accidental deletion, hardware failure, ransomware attacks, and other occurrences by using robust backup solutions.

Security Audit and Assessments: Security audits and assessments are conducted on a regular basis to evaluate the efficiency of security controls, find vulnerabilities, and suggest better enhancements. These evaluations assist businesses in maintaining a robust security posture.

Cloud security: Keeping virtual resources safe in the cloud has became increasingly important as more and more businesses are using it. Secure

configuration, data protection, identity and access management, and encryption are all components of cloud security, that work towards protecting data, applications, and infrastructure in cloud environments.

Malware protection: Virtual security is seriously harmed by malicious programs like ransomware, worms, viruses, and Trojans. To handle these malware, antivirus and anti-malware softwares are essential. They identify and block the known and unknown threats right away.

Adversarial machine learning: Adversarial machine learning [2] is the study of understanding and protecting machine learning models from malicious attackers by providing adversarial samples with the training input. The model identifies the data patterns and makes much accurate predictions over time making our AI model much robust.

In this project, we specifically focused on adversarial machine learning and malware detection.

Virtual security is a complex and developing field and works on providing ways to address and deal with current or upcoming dangers. In order to maintain a strong and solid virtual climate, we need to stay aware of rising innovations and should remain informed about the most recent security rehearses.

1.1 Overview of Malware

In this section, we shall discuss about malware and its purposes. A program designed to interrupt the normal functioning of a computer system, malware is a blanket term for all viruses, worms, trojans and various other malicious programs.

1.1.1 Malware Definition

The unauthorised access to a computer system, network or device with an intention of exploiting, stealing or manipulating its data is referred to as malicious softwares or malware. Its intention is to harm, damage and gain private and sensitive information over a system or network without user's permission. Mostly used by cyber frauds for a number of reasons like getting control over multiple systems to launch denial-of-services attacks, infecting systems, identify or personal data theft.

1.1.2 Purposes Of Malware

The primary objective of malware is to disturb, harm, or deal with computer systems or take sensitive data.There are multiple ways in which a malware can enter into your system, for example via email attachments, malicious links or websites, software downloads, infected USB drives, and even exploiting vulnerabilities in applications are all.
Depending on its type and purpose, malware can perform a variety of harmful activities once it has infected a device or network. Following are some typical acts of malware:

- ***Data theft***: Malware can stole your personal information, login details, identity theft, credit card details or any other confidential data.
- ***System disruption***: Malware can infect your system making it inoperable or slow or performing certain actions that could be annoying and unsafe.
- ***Unauthorised access***: Malwares might give unauthorised access to attackers which in turn perform malicious exercises.

- ***Attacks by Trojans***: Malware of this kind disguises itself as a desirable program to get into the users' system and exploits the vulnerabilities of the system in order to gain access.

- ***Spyware and ads***: Malware can spy over users activities and steal the data that could be important, display unnecessary advertisements or direct you to malicious websites.

1.1.3 Classification Of Malware

Malware can take many different forms and use many different strategies to accomplish its goals. Some typical types of malware are as follows:

1. ***Viruses***: These are self-replicating programs that attach themselves to real data and spread by affecting other documents. Viruses can clear or erase data, render the system useless or spread to various devices.

2. ***Worms***: Worms are independent that regenerate themselves and spread across devices or networks without requiring any assistance by the user. Some of the worm families are:

    - ***Vobfus***[4]: Vobfus is a worm family that creates multiple copies of itself on the computer systems that it searches on a network. Once the user runs it on the computer, the program is activated.
    - ***Mydoom***[5]: Mydoom is one of the fastest- spreading worm family that affects the Windows operating system. It spreads via email attachments. After the user clicks on these infected email attachments, it infects the operating system and sends malicious emails to the users' contacts.

[4]https://threats.kaspersky.com/en/threat/Worm.Win32.Vobfus/
[5]https://nordvpn.com/blog/mydoom-virus/

- ***Sasser***[6]: Sasser is a worm that infects Windows operating system and exploits its vulnerabilities to spread across a network causing unstability.
- ***Code Red***[7]: Code Red is a worm that targets the systems having Windows operating system with Microsoft IIS installed.
- ***Slammer***[8]: A slammer is a worm that targets microsoft SQL server causing denial-of-services conditions.

3. ***Trojans***: Trojans, also known as Trojan horses, imitate actual softwares or files tricking the users into running them. Once the user open these files or download such software it spreads into the device and starts performing malicious activities. Some of the trojan malware families are listed below:
   - ***Adload***[9]: Adload is a trojan that affects MacOS platforms and recruits malicious programs like adware and Potentially Unwanted Applications(PUA).
   - ***Renos***[10]: Renos is a Trojan that insinuate itself into the computer system by faking security warnings. It displays annoying security concerns and tricks the user into downloading a third- party cleaning software.
   - ***Startpage***[11]: Startpage is a Trojan family that alters and changes the homepage or startpage of the internet browser without consent

[6]https://www.techslang.com/definition/what-is-the-sasser-worm/
[7]https://www.kaspersky.com/blog/history-lessons-code-red/45082/
[8]https://cyberhoot.com/cybrary/sql-slammer-virus/
[9]https://www.bleepingcomputer.com/news/apple/new-adload-malware-variant-slips-through-apples-xprotect-defenses/
[10]https://www.enigmasoftware.com/trojanrenos-removal/
[11]https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/StartpagethreatId=15435

that redirects the user to malicious sites.

- ***Winwebsec***[12]:This is a Trojan family that pretends to scan your system and then lists a numbers of infections. It will trick the user into thinking that their system has been infected, displays security warnings whenever normal applications are tried to run and demands that it can only be fixed if you make a purchase,
- ***Zeroaccess***[13]: Zeroaccess is a Trojan that affects the Windows operating system by downloading other malware program files into the infected system, while itself remains hidden.
- ***Dridex***[14]: Dridex is a banking trojan that is used for financial frauds and targets victim's banking information. It aims to gain the victim's account details and credentials in order to gain access to their assests.

4. ***Ransomware***: A client's system or files that are seized by ransomware, are made inaccessible until they are provided with the ransom.It spreads via emails, malicious sites or links and takes advantage of user's helplessness. Some of the ransomware families are:

   - ***CryptoLocker***: CryptoLocker is a ransomware that gains access to the system files and encryptes them. It then demands ransome in order to decrypt the files.
   - ***Ryuk***[15]: Ryuk is a ransomware that has affected various organisations like healthcare, academics, government and industries. It

[12]https://www.microsoft.com/en-us/wdsi/threats/threat-search?query=win32/winwebsec
[13]https://www.cybereason.com/blog/what-is-zeroaccess-malware
[14]https://www.techtarget.com/searchsecurity/definition/Dridex-malware
[15]https://www.trendmicro.com/en$_i n/what-is/ransomware/ryuk-ransomware.html$

can encrypt files remotely and is also capable of Wake-On-Lan, which wakes computers for encryption.

- ***GandCrab***[16]: It is a widespread Ransomware-as-a-service offering, that encrypts files and demands ransome to decrypt.
- ***WannaCry***[17]: WannaCry is a ransomware that exploits the vulnerabilities in windows systems and gains access or encrypts the files. This ransomware caused a global disruption in the year 2017.

5. ***Spyware***: Spyware is intended to spy on data without the client's permission. It might It screens exercises, for example, keystrokes, web perusing propensities, and login qualifications, and sends the gathered information to the assailant. Spyware is frequently utilized for unapproved reconnaissance, data fraud, or designated assaults.

6. ***Adware***: Adware's job is to frequently show undesirable ads or notices that helps produce income to the attacker. It can be found installed in actual softwares or can directly be displayed in the form of advertisements in websites. While it might not cause much harm as compared to other malwares, it can be annoying and troublesome for the clients and might affect clients security.

7. ***Keyloggers***: Keyloggers are used to steal sensitive information like login details, passwords, credit card information to gain unauthorised access to users accounts.

8. ***Rootkits***: Rootkits are used to conceal malwares or other malicious activities from recognition by antivirus programs. They gain access to

[16]https://www.malwarebytes.com/gandcrab
[17]https://www.malwarebytes.com/wannacry

the device's operating system and make changes to files and processes, making detection and defense tough.

1.2 Introduction to Adversarial Machine Learning

Machine learning describes how computers can be programmed to identify patterns in data and make much accurate predictions over time. A problem with machine learning is that while the model identifies data patterns, attackers could alter the training data with false information or we can say, adversarial examples. This could trick the model into producing incorrect outputs. Researches[18] show that just by adding a few pieces of tape, a self-driving car can misclassify a stop sign as a speed limit sign.

Adversarial machine learning is a field of study that deals with detecting and protecting machine learning models against malicious attacks. It focuses on the methods to create robust machine learning models that are immune to any sort of malware attack.

S. Asha and P. Vinod [1] while illustrating the AML architecture (figure 1.3) suggest that machine learning models consists of two phases, in which they operate, the training phase and the testing phase. In the first phase i.e. the training phase, the machine learning model is trained to identify and produce output of the label class. After the model is trained, adversarial samples are created in order to evaluate the performance of the proposed model and its efficiency. This is called the testing phase and the model is considered to be robust if it effectively detects adversarial examples and handle them.

[18]https://www.bleepingcomputer.com/news/security/you-can-trick-self-driving-cars-by-defacing-street-signs/

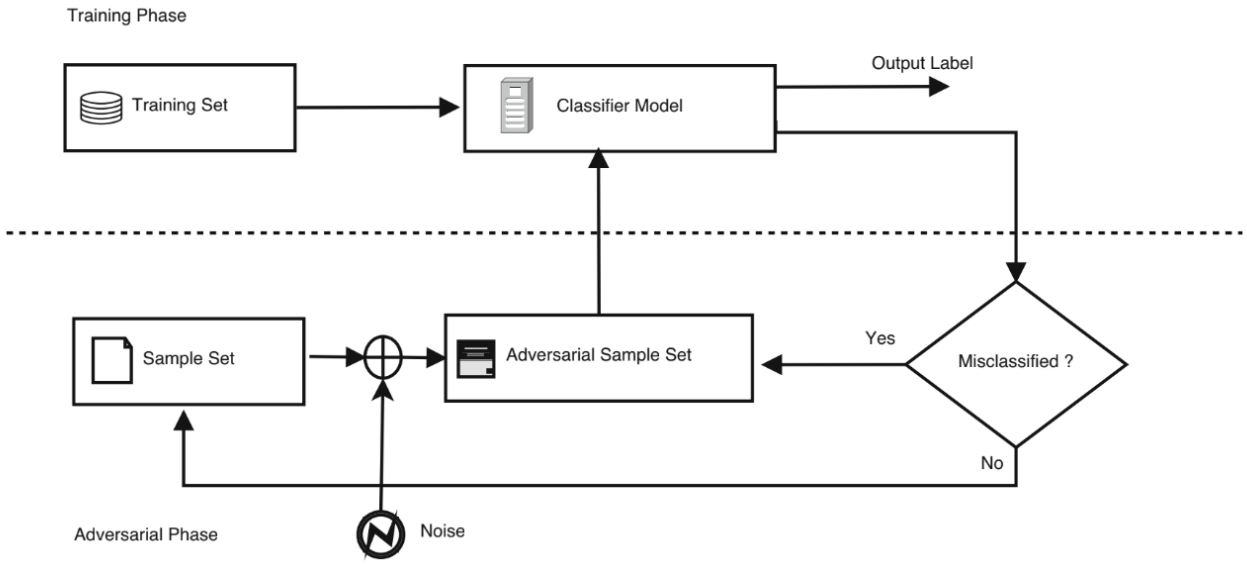

Figure 1.3: Basic Architecture of AML [1]

1.2.1 Techniques

Various adversarial robustness methods have been developed by the researchers to secure machine learning models against adversarial attacks:

Adversarial training: By including adversarial examples we enlarge the training set, which adds to the training process. By presenting the model these adversarial examples it can figure out how to be more strong and robust against such attacks. Kurakin et al. [3] experimented with adversarial training using various types of single step methods. They suggested that adversarial training should be carried out in two scenarios:

- When a model is so complex that it begins to include noise alongwith the training data, and there is a requirement of a solution to resolve such overfitting problem.
- When security is a concern against adversarial examples. In such case,

adversarial training provides better results as compared to any other defense while only reducing accuracy by a small amount.

Defensive distillation: By altering the output of a model that has already been trained, this method trains a model to be resistant to adversarial attacks. Papernot et al. [4] describes Distillation as a method of training designed to train a Deep Neural Network (DNN) with the help of the knowledge that was transferred to it from another DNN model. The basic objective behind this transfer of knowledge through distillation is to solve the problem of complexity while computation of these models by transferring knowledge from larger architectures to smaller ones.

Feature squeezing [5]: The details of some features in the input data are intentionally reduced, known as feature squeezing, which helps reducing the search space for possibly present adversarial examples. An attacker may find it more challenging to locate effective perturbations by employing this strategy.

Adversarial detection [6]: In this method, algorithms are created to identify and highlight possible adversarial examples that might be present during the training phase of the model. The model is able to handle the potential risks that can be caused by the adversarial examples present in the training input data.

1.2.2 Classification Of AML Attacks

Based on the type of vulnerabilities in the training data, AML attacks can be classified into:

- ***Evasion Attacks*** [1]: In evasion attacks, attacker might introduce data to intentionally deceive an already trained model into making

errors. The data that the model gains through heuristic approach by learning from its environment is altered by the attackers. It results in exploiting the knowledge that was already present in the classifier, causing security breach.

- ***Poisoning Attacks*** [7]: In poisoning attacks, attackers might disrupt a machine learning model by altering or poisoning the model while it is learning by introducing training data that is incorrect or misleading.

Based on the type of target class, an AML attack can be:

- ***Targeted Attacks*** [8]: Targeted attacks aim to cause classification based on a target class function. The network misclassifies data and labels it into a specific class.

- ***Untargeted Attacks*** [9]: In an untargeted attack, a specific class is not targeted. The input is manipulated such that the response of anything but labelled.

On the basis of information an attacker might posses, attacks can be divided into these categories:

- ***White box attack*** [10]: In such attacks, attackers have access to the model's parameters, architecture and features. They also have the access to training dataset.

- ***Black box attack*** [11]: In these attacks, it is assumed that the attacker has no knowledge about the model. Neither does the attacker can access and infiltrate the training dataset.

S. Asha and P. Vinod [1] also described *grey box attacks* in their paper, in which the adversory has some knowledge about the model.

1.3 Adversarial Machine Learning Tools

In this section we mentioned the various tools that we have considered during our research:

1. ***Cleverhans*** [12]: It is a software library that provides standardised references to implementation of techniques for generation of adversarial examples and also adversarial training. The cleverhans library is intended to use for two purposes:

    - To construct a machine learning model that is robust with the help of adversarial training methods by the developers.
    - Encouraging researchers to report accuracy of their proposed models in the adversarial setting.

2. ***Adversarial robustness toolbox*** [13]: ART is an open source python library that helps in the research and development field to handle machine learning models against adversarial attacks in order to make them robust and more secure. It provides with interfaces for classifiers of TensorFlow, PyTorch, MXNet, XGBoost, LightGBM, CatBoost, and GPy, which are some of the most popular machine learning libraries.

3. ***Foolbox*** [14]: Foolbox is a Python package used for generating adversarial examples in order to analyze and compare the machine learning models on the basis of their robustness. The idea behind it is that the lowest perturbation required for generating an adversarial example is the most comparable robustness measure.

4. ***AdvBox*** [15]: Advbox, based on python and uses object oriented programming, is a toolbox which is used to construct adversarial

examples in order to trick neural networks in popular machine learning libraries like PaddlePaddle, PyTorch, Caffe2, MxNet, TensorFlow. It also provides a basis for the robustness of machine learning models.

5. ***DEEPSEC*** [16]: It is a uniform platform for deep learning models that aims to build robust machine learning models against adversarial attacks. Currently, DEEPSEC consists of 16 state-of-the-art attacks and 13 state-of-the-art defenses. DEEPSEC is the first platform that can be used to measure how vulnerable the deep learning models are and to analyse how much effective are the attacks and defenses. It also allows to carry comparative analysis on attacks or defenses for informational purposes.

6. ***AdverTorch*** [17]: Advertorch is a python toolbox used for adversarial robustness research. A number of attack implementations, defense and robust training method implementations are included. It is built on PyTorch, and provides with advantages for dynamic computational graph to provide effective reference implementations.

Table 1.1 shows the Machine learning libraries supported by the six tools- Adversarial Robusteness Tool (ART), FoolBox, AdvBox, Cleverhans, AdverTorch and Deepsec. ART is compatible with 10 Machine Learning libraries. Foolbox supports 5 libraries and AdvBox supports 6 machine learning libraries. CleverHans, for the time being, is fully compatible with three machine learning libraries PyTorch, JAX and TF2. AdverTorch is currently based on PyTorch, and Deepsec is based on PyTorch, Torchvision and Numpy.

Table 1.1: Compatibility of AML Robustness Tools

Libraries	**Tools**					
	ART	**FoolBox**	**AdvBox**	**Cleverhans**	**AdverTorch**	**Deepsec**
Tensorflow	✓	✓	✓			
Keras	✓	✓	✓			
JAX		✓		✓		
PyTorch	✓	✓	✓	✓	✓	✓
PaddlePaddle			✓			
Torchvision						✓
TF2				✓		
MXNet	✓		✓			
Scikit-learn	✓					
XGBoost	✓					
LightGBM	✓					
CatBoost	✓					
GPy	✓					
Tessaract	✓					
NumPy		✓				✓
Caffe2			✓			

1.4 Motivation

It has been discovered that the security of the majority of systems is at major risk, despite the numerous methods and strategies that have been discovered over time to prevent security breaches and the operation of malicious programs.

On November 23,2022 the All India Institute of Medical Sciences (AIIMS) in Delhi allegedly experienced a cyberattack[19] that rendered its servers inoperable. The most recent cyberattack compromised five servers of the All India Institute of Medical Sciences (AIIMS), encrypting an estimated 1.3 terabytes of data. A news report expressed that as the result of this cyberattack the records of crores of patients were compromised.

SOVA , a mobile banking Trojan virus[20] has been targeting and posing threat

[19]https://thewire.in/government/aiims-servers-cyberattack-ransomware-rajya-sabha

[20]https://www.business-standard.com/article/technology/new-mobile-banking-trojan-virus-prowling-in-indian-cyberspace-warns-govt-122091500534$_1$.*html*

to Indian customers as it encrypts the android mobiles for ransome. While it extracts passwords and usernames from users' system, it is really hard to uninstall it from the device.

A report[21] stated, "Malware attacks rise in 2022". According to the annual security study conducted by Trend Micro, India experienced approximately 700,000 malware attacks in 2022, up from 650,000 in 2021.

CloudSEK, a cyber intelligence company, wrote in a report[22] that YouTube has recently seen an increase in videos with harmful links to infostealers in their descriptions. AI-generated personas are used in many of these videos to trick viewers into trusting them. CloudSEK says that people are using AI-generated videos more and more for good things like education, recruitment, and promotion. But cybercriminals are also using this technology for malicious purposes.

One of the reasons we decided to work on this project is that when we did our research and tried to find software or tools that worked in this area, we found that the majority of researchers had used machine learning. However, very few people, if at all, have utilized deep learning. Due to the issue of overfitting, machine learning strategies cannot produce an effective result in real-world scenarios with extremely large datasets to analyze.

1.5 Problem Statement

To design and implement adversarial machine learning framework in virtual environment.

The problem statement can be further classified into the following objectives:

[21]https://economictimes.indiatimes.com/tech/technology/ransomware-malware-attacks-rise-in-2022-report/articleshow/99094491.cms?from=mdr

[22]https://economictimes.indiatimes.com/news/international/us/ai-generated-youtube-videos-spreading-info-stealing-malware-heres-how/articleshow/98639692.cms?from=mdr

- To design and implement robust malware detection model
- To generate adversarial sample using various open source AML tools.
- To design and implement transfer learning to retrain the design model to detect adversarial sample.
- To validate the model using publicly available dataset.

1.6 Organisation of The Project

Beginning with a brief overview of virtual security, we move on to an in-depth look at the behaviour of various types of malware, adversarial machine learning, and our motivation for working on this project. We have attempted to feature every one of the insights regarding virtual security, malware and adversarial machine learning.

In chapter two, we review the relevant literature examining recent advancements in the successful detection and creation of robust machine learning models. Adversarial training, defense mechanisms, GANs, certifiable robustness, ensemble methods have been explored to improve robustness of machine learning models to handle adversarial attacks.

In chapter three, we discuss the architecture of the proposed robust model in detail. Discussing the three phases of the machine learning model to handle adversarial attacks: pre-processing and feature extraction, training of the pre-processed data, clustering of data, adversarial defence mechanisms and retraining of data.

In chapter four, we shared the results of the experiments and comparisons that we have performed in our project. We examined various datasets with the help of multiple ML and DL models.

In the last chapter of our project, we briefly conclude everything in our research and talk about future work possibilities in malware detection.

Considerable number of references and various sources have been used in our research which provides a broad view of accessible methods making it simpler to compare our work logically to others.

Chapter 2

Literature Review

Machine learning has a significant impact on virtual environments, revolutionizing various aspects of virtualization operations, management, and security. However, these innovations and transformations come with potential risks. It has became a major concern today that how vulnerable are these machine learning models. By adding incorrect or misleading data at the input, adversarial attacks aims at taking advantage of the model's weaknesses, which tricks these machine learning models into making errors.

2.1 Adversarial detection

Adversarial detection aims at developing techniques and algorithms to detect and identify adversarial examples or attacks in machine learning models. Adversarial examples are carefully crafted inputs that are designed to deceive machine learning models and cause them to produce incorrect or unexpected outputs. These attacks aim to exploit vulnerabilities in the models and can have significant implications in cyber security. The concept of adversarial detection revolves around developing robust defenses

and countermeasures against adversarial attacks. This involves studying and analyzing the characteristics of adversarial examples, understanding the underlying vulnerabilities in machine learning models, and designing detection mechanisms that can accurately identify and classify adversarial inputs.

Grosse et al. [18] proposed on the basis of a theory that adversarial examples are most likely to be found at the low density areas of the input. With the help of Generative models or density estimation models, adversarial examples can be identified and rejected before having any impact on the model's predictions by simply estimating the input density. The authors determined that real examples and adversarial examples possess distinct statistical properties that can be utilized for detection. Adversarial examples can be identified by using a factual test, the two-sample Kolmogorov-Smirnov (KS) test, even when the attacked model continues to misclassify them. Even under adaptive attacks, the detection method remains effective, making it a promising defense mechanism. The two-sample KS test and how it resists to the adaptive attacks are used to demonstrate the efficiency of the proposed method.

Gaur et al. [19] proposed a malware detection approach based on Deep Neural network (DNN), known as DeepHyperv, in order to detect malware attacks in virtual environment by performing deep virtual memory analysis. In this method,initially logs are generated by performing memory introspection on the proposed hypervisor. Then these logs are already processed and for every malware or benign practice, feature vectors are produced. Lastly, the models for classification of training and testing process the data. The accuracy of

about 91 per cent with False Positive rate (FPR) of about 8.15 per cent was given by DNN.

Zhang et al. [20] proposed a soft relevance value (s -value), to determine the feature soft relevance that assess classified results using the mixed distance criterion in order to recognise research models that were not labelled in the training collection as a new family. It determines to which of the original families does the malware belongs, using the s-value. XGBoost and Microsoft Malware classification challenge dataset that consists 9 distinct families of malware is used. The results showed about 12 hours training time while only about 0.5 seconds prediction time. The accuracy of classifying malware resulted about 99.8 percent.

L. Zheng and J. Zhang [21] presented a malware detection method in cloud service platforms without causing any kind of harm to the clients privacy. Using Visualised Memory Change Area Dimensionality Reduction (VMCADR), on a binary memory snapshot malware detection is carried out directly without accessing the user system data. Including about 3300 memory snapshots, the authors collected a large amount of real malware samples and utilised Xen environment. VMCADR has a remarkable malware detection rate, however, it still has problems with reducing delays which might affect the clients.

Qiang et al. [22] proposed a new robust malware detection model with dynamic analysis using control flow traces. In addition, a malware classifier is also proposed by converting the control flow traces into sequences of bytes and further applying Convolutional Neural Networks (CNN) and Long Short

Term Memory (LSTM). The proposed classifier shows a good performance with an malware detection accuracy of about 95 percent and unseen malware detection accuracy of about 94 percent. The overall accuracy of about 83 percent.

Table 2.1: Related work

Author	Year	Detection	Approach	Key Algorithm
Grosse et al. [18]	2017	Static	Kolmogorov Smirnov test	Generative or density estimation models
Darshan et al. [23]	2019	Dynamic	Hybrid Features based Malware Detection System	Linear Support Vector Classification
Amer et al. [24]	2020	Dynamic	Malware API call graph	Word embedding, Markov chain
Zheng and [21] Zhang	2022	Static	VMCADR	SNN
Zhang et al. [20]	2022	Dynamic	Soft relevance evaluation	Classification model based on XGBoost
Rizvi et al. [25]	2022	Static	Unsupervised clustering and features attention-based DNN	PROUD-MAL
Ragaventhiran [26] et al.	2022	Dynamic	windows-based system call sequence	LSTM auto encoder
Quieng et al. [22]	2022	Dynamic	DNN	Control Flow traces, CNN, LSTM
Gaur et al. [19]	2023	Dynamic	DNN	Deephyperv
Yousuf et al. [27]	2023	Static	Portable executable (PE) malware detection	Information Gain and Principal Component Analysis
Buriro et al. [28]	2023	Dynamic	Portable executable (PE) malware detection	MALWDC
Chaganti et al. [29]	2023	Dynamic	Portable executable (PE) malware detection	CNN

2.2 Adversarial attacks and defenses

Szegedy et al. [30] demonstrated that even minor changes in the input data can misclassify machine learning models significantly, introducing the concept of adversarial attacks. They found that adversarial examples can cause neural networks to wrongly classify inputs with high confidence. This challenged the capacity and robustness of neural systems. The paper further examines the adaptability of adversarial models across various neural network structures showing that even when these models were having distinct architectures and

were trained on different datasets, the adversarial examples generated for one network architecture can trick other architectures as well.The research presented in this paper reveals the strange properties of neural organisations. The results made us realise the importance of understanding the weaknesses and limitations of these models. These findings provoked further research into adversarial attacks and defenses, which has let to introduction of architectures that are more secure and robust.

Goodfellow et al. [31] further divided adversarial attacks into poisoning attacks, where attackers might attack or disrupt a machine learning model by poisoning or contaminating the model by introducing input that is inaccurate or misleading. The "Fast Gradient Sign Method" (FGSM) is then suggested to generate adversarial examples effectively in the paper. To determine the disturbances with the greatest loss, FGSM makes use of the loss function's gradients in relation to the input data. The presence of adversarial examples is suggested to be explained in two ways by the authors: geometry with high dimensions and linearity. They suggest that despite being a simplification of the primary data distribution, linearity can make sense of the presence of adversarial models. They also suggest that the concentration of adversarial examples near decision boundaries is caused in high dimensional space to place the data by high-dimensional geometry.

Adversarial training is one way to deal with adversarial attacks, we enlarge the training set by including adversarial examples which adds to the training process. By presenting the model these adversarial examples it can figure out how to be more strong and robust to such attacks.

Madry et al. [32] proposed a robust optimisation framework for adversarial training, achieving robustness against a number of attacks. To make the models immune to adversarial attacks, they proposed a preparation approach called Robust Enhancement. Worst-case disturbances are the principle of this strategy during the training process. By improving the model to reduce the worst case loss, the resulting model becomes more resistant to adversarial attacks. The paper presents another calculation, called Projected Gradient Descent, used to track the worst scenario for each contribution during training.

Papernot et al. [4] introduced defensive distillation, which involves altering the output of a model that has already been trained. Concerning adversarial protection, refining includes preparing a teacher model, which is a robust deep neural network, and a student model, which is a modest and less complicated network that expects to follow the teacher's way of behaving. The central idea is that the predictions of the teacher model serve as soft targets for training the student model. The authors also mentioned the drawbacks and benefits of distillation as a defense strategy. They observed that, despite the fact that distillation can increase robustness, it might not completely protect the model against all kinds of attacks.

However, later research carried out by Carlini and Wagner [33] revealed that defensive distillation is insufficient to protect the machine learning model against powerful adversarial attacks on its own. The authors demonstrate that adversarial examples generated against the teacher model can be easily transferred to the student model, resulting in misclassifications and reduced robustness. A new attack strategy optimization-based attack, that

specifically targets defensive distillation, was the subject of the paper. This attack strategy takes advantage of the way that protective refining depends on probabilities, which can be used to figure out an improvement issue that meets the limitations of the understudy model.

S. Asha and P. Vinod [1] focused on analysis of distinct adversarial machine learning tools that would help secure AI models against adversarial attacks by effectively detecting and preventing them The tools used: CleverHans, Foolbox, AdvBox, DEEPSEC, Adversarial Robustness Toolbox, and Adver-Torch. Ensemble methods, input pre-processing and adversarial training were investigated as defense- strategy based tools. In order to analyse and evaluate the robustness of the AML tools, Cifar-10, ImageNet and Modified National Institute of Standards and Technology database(MNIST) datasets were used. Each tool was evaluated with six kinds of adversarial attacks under different datasets. ResNet, InceptionV3 and Madry were used by the machine learning models. Cifar-10 dataset was used for ResNet, ImageNet dataset for InceptionV3 and the MNIST dataset was used for Madry.With a support of 29 attack scenarios, Foolbox provided the maximum attack implementation. Providing the highest defense strategy implementation, in ART 15 distinct defense mechanisms are supported. Under all attack conditions, the effectiveness of attack was higher for CleverHans tool. It was recorded that the overall attack accuracy of cleverhans was 0.74 and 0.126 was the recorded perturbation rate.

Carlini et al. [34] discussed the methodological foundation, reviewed practices that are commonly accepted and suggested new defensive methods against adversarial examples. It also provides a detailed description of the

common weaknesses and in addition recommended on avoiding incorrect and misleading reported results while working with adversarial robustness.

Yan et al. [35] particularly focused on malware classification and provided an overview of adversarial attacks and defense strategies. Examining the increasing danger of malware and describing the importance of successful strategies of malware classification to identify and deal with these malicious programs. Machine learning based classifers in white-box and black-box attack settings were looked closely. Various defense methods were also examined which could work against adversarial attacks by improvising the robustness of malware classifiers.

Chapter 3

Proposed Methodology

This chapter describe about the proposed methodology, datasets and experiments used to evaluate the attacks effectiveness. Building a robust module for detecting malware consist of various methodologies.

According to our experiment Machine Learning Model to handle adversarial attacks consist of three modules:

Malware Detection Module: *This modules includes preprocessing of data,training and testing of the data to train and build our model.*

Attack on Malware Detection Module: *This modules includes clustering of data family vise and adversarial attack on data to misclassify our model.*

Retrained Malware Detection Module: *This modules includes the retraining of the model with adverserial data to make our model robust and efficient against adversarial attacks.*

The execution flow of these design components has been shown in 3.1.

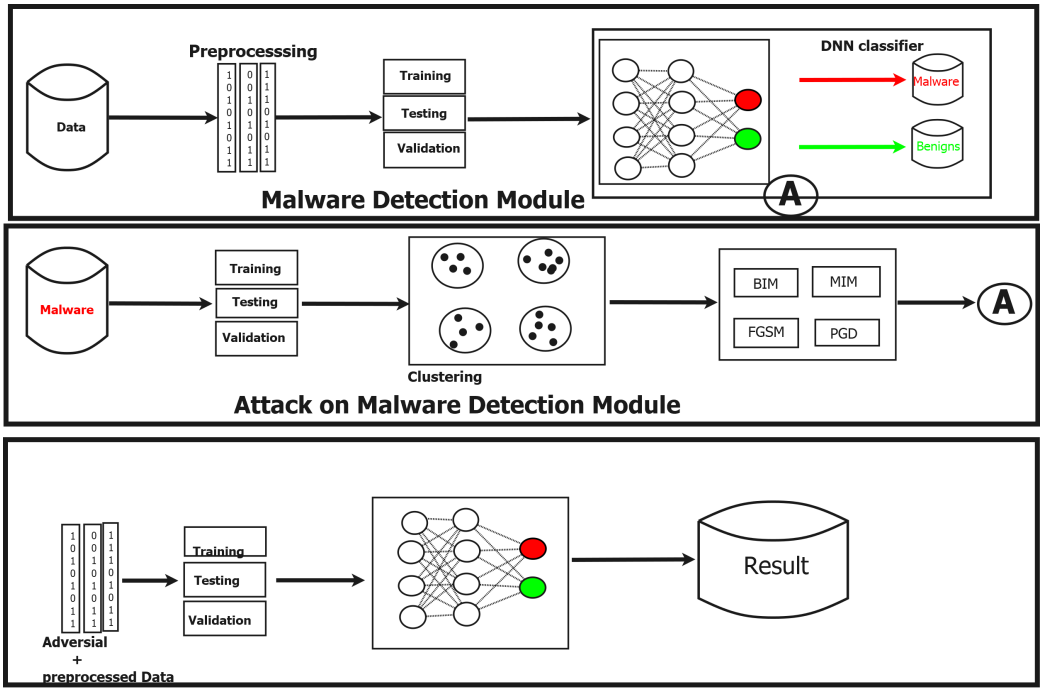

Figure 3.1: Architecture of Proposed Robust Model

3.1 Malware Detection Module

Dataset: A dataset is a collection of structured or unstructured data that is organized and stored together in order to perform some specific tasks. For evaluating the robustness of AML tools, the dataset contains different types of malware and their families.

The provided dataset is than preproccesed which is an essential step in building a robust model for malware detection. It involves cleaning, transforming, and thus preparing the data to make it suitable for the machine learning algorithms to analysis it.

Data Cleaning involves to identify and handle the missing data in the dataset. This can be done by inputing the missing values or removing the samples

containing the missing values.

Data Transformation involves normalization or scaling of data to a similar range with techniques like standardization and normalization. It helps to handle the imbalance data in order to prevent the model from being biased towards the majority class.

After the preprocessing of the dataset choose an appropriate machine learning model for malware detection model. Commonly used models are decision trees, random forests, support vector machines (SVM), neural networks any many more.

The dataset is than splited into training and testing sets in ratio of 80-20. The larger dataset is used to train the data while the smaller dataset to used to test our trained model. For our experiment we opted for the Deep Neural Network(DNN) due to its impressive performance and versatility.Additionally, DNNs are known for their ability to handle missing values, which are common challenges in datasets. They can automatically learn relevant features from the data, reducing the need for manual feature engineering. DNNs can also leverage large amounts of data for training, which is often available in applications.

A DNN is a type of artificial neural network that consists of multiple layers of interconnected nodes, known as neurons. Each neuron takes inputs, performs a mathematical operation on them, and produces an output.DNN consist of 3 layers.

1. *Input Layer:* The DNN module starts with an input layer, which receives the initial input data. Each neuron in the input layer represents a feature of the input data[1].

[1]https://towardsdatascience.com/everything-you-need-to-know-about-neural-

2. *Hidden Layers:* After the input layer there are mutiples hidden layers present in DNN model. After receiving input from the the input layer multiple neurons in each hidden layers apply a mathematical transformation, such as a weighted sum and an activation function, to the inputs. This transformation helps to extract complex features s from the input data[2].

3. *Output Layer*: The output of the hiddens layers act as input for output layer, which produces the final outputs of the DNN module. The number of neurons in the output layer depends on the specific task the DNN is designed for[3].

After running the model, the classification report and confusion matrix are used to evaluate and analyze the performance of the model. The classification report includes metrics such as precision, recall, F1-score, and support for each class. The model can detect whether a sample is benign or malicious. In the case of malicious samples, it can further classify them into different malware families based on their characteristics.

3.2 Attack on Malware Detection Module

To perform attacks, we first cluster our data according to the malware families.

Clustering: Clustering is the technique used in machine learning which is

networks-and-backpropagation-machine-learning-made-easy

[2]https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy

[3]https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy

used to group the data with similar characteristics together. Which will find the natural groupings in the trained data.

Use of Clustering : Clustering is used in preprocessed data to group similar malware families together and reducing its dimensions.

Then, we implement the silhouette method to determine the best number of clusters suitable for each family.

> ***Sillhoute Method:*** This method is very useful to find the value of k i.e, no of cluster when the eblow method don't show any point[4].
>
> The value of sillhoute range from -1 to 1:
>
> **-1 : wrongly assigned in cluster**
>
> **0 :Overlapping wrongly assigned in cluster**
>
> **1 : Perfectly assigned in cluster and sample is far away from the neighboring clusters.**

After clustering the data according to malware family. We found the best number of cluster suited per family to provide and clustering is done using K-Mean algorithm. After defining the number of clusters in the algorithm it will form the number of clusters for each family.

Then, we implemented techniques such as adversarial attacks or manipulating features to confuse the model's decision-making process. These techniques aim to deceive the model and make it classify data incorrectly.Since, our goal is to misclassify the data by providing miscellaneous data.

Adversarial attacks mechanisms: The clustered data is sequentially provided to the prepared attack strategy for data manipulation. After manipulation, the data is passed through the model. The effectiveness of

[4]https://scikit-learn.org/stable/auto$_e$*xamples/cluster/plot$_k$means$_s$ilhouette$_a$nalysis.html*

the attack is validated by checking if the model misclassifies the manipulated data.

Since we are using a DNN model, we have utilized the CleverHans AML (Adversarial Machine Learning) library, which offers a range of evasion attacks. Some of the evasion attacks provided by CleverHans include:

1. **Fast Gradient Sign Method (FGSM):** The Fast Gradient Sign Method create multiple adversial attack example by using the gradient of neural network gradients.The mathematical expression for FGSM attack is[5]:

$$\text{adv\_x=x} + \epsilon * (sign(\bigtriangledown_x J(\theta, x, y))$$

2. **Basic Iterative Method (BIM):** Adversarial Examples in the Physical World's Basic Iterative Method (BIM)[6] is a straightforward extension of the Fast Gradient Sign Method. It applies FGSM to an image multiple times with step size, or the change in pixel value per iteration, rather than in one large step.

3. **Momentum Iterative Method (MIM):** The momentum iterative techniques(MIM)[7] can be effectively generalized to other attack settings. Any iterative method can be extended to its momentum variant by substituting the accumulated gradient of all previous steps for the current gradient.

4. **Projected Gradient Descent (PGD):** PGD[8] is a standard way to solve

[5]https://towardsdatascience.com/perhaps-the-simplest-introduction-of-adversarial-examples-ever-c0839a759b8d

[6]https://www.neuralception.com/adversarialexamples-bim

[7]https://openaccess.thecvf.com/content$_c vpr_2$018/$papers/Dong_B oosting_A dversarial_A ttacks_C VPR_2$018$_p aper.p$

[8]https://angms.science/doc/CVX/CVX$_P GD.pdf$

constrained optimization problem.A constraint set starting from a initial point PGD iterates the following equation until a stopping condition is met:

$$\mathrm{x}_{k+1} = P_Q(x_k - \alpha_k \bigtriangledown f(x_k))$$

These attacks are applied to manipulate the input data in order to deceive the DNN model's classification.

3.3 Retrained Malware Detection Module

The process of retraining the model with adversarial data is a application of transfer learning, Where it aims to improve the effectiveness and efficiency of the model by leveraging pre-existing knowledge.

When it comes to enhancing the model's robustness against adversarial attacks, the process of retraining the model with adversarial data is commonly referred to as adversarial training. Adversarial training involves training the model on a combination of clean and adversarially perturbed examples to improve its resilience against adversarial attacks. It aims to make the model more effective and robust in the presence of adversarial inputs.

After conducting the attacks on the data clusters,adversarial training techniques are deployed to enhance the classifier's robustness.

In our defensive strategy, we store the misclassified data and retrain the model by including both the misclassified data and the original training data. By incorporating the misclassified samples into the training process, we aim to improve the classifier's ability to correctly classify similar instances in the future.

Chapter 4

Experiments and Results

The prototype of the proposed PROJECT has been implemented in the test bed setup of our laboratory.There are five physical machines on which experiments have been performed, and their configuration is as follows: 8GB RAM, core i7 processor, 500GB SSD, and ubuntu 20.04.05 LTS. Each of the physical machine runs Xen hypervisor and on the top of which two Windows VMs are created. A malware analysis environment has been created using these machines to generate malware execution logs at the hypervisor. Various open-source libraries/tools, such as DRAKVUF [7], LibVMI [19], volatility, have been installed in all the VMs of each physical machine. To pre-process and analysis huge logs (in terabytes), a good configuration server having 32 GB RAM, 500 GB SSD and ubuntu 22.04.1 LTS (guest OS) and an Intel® Xeon® processor, is deployed in lab. At software level, anaconda based python programming framework has been installed on server to pre-process the logs and analyse them using machine learning models for classification.

Based on the evaluation of the data sets (system call, callback, DLL, process, and registry), we use single datasets to analyze them using multiple models such as decision tree, random forest, support vector machine, k-nearest

neighbors, deep neural networks, etc.

Table 4.1: Accuracy of models on different features

accuracy							
	dt	random forest	svm	knn	gradient boosting	log	**DNN**
dll	87.93%	98.53%	83.12%	99.67%	99.91%	99.02%	**99.44%**
filename	99.81%	93.25%	78.57%	99.21%	99.86%	66.20%	**97.02%**
sucess_rate	91.41%	98.41%	78.92%	80.09%	71.16%	77.30%	**97.95%**
system_call	87.98%	97.01%	81.20%	76.86%	75.50%	74.11%	**97.67%**
time_deviation	92.85%	97.33%	86.36%	84.49%	84.68%	78.60%	**94.41%**

Table 4.2: TPR, TNR, FPR, TNR values for DNN model

	DNN			
	TPR	FNR	FPR	TNR
dll	99.264245	0.735755	0.091969	99.908031
file name	96.130722	3.869278	0.48366	99.51634
suceess rate	97.77202	2.22798	0.278498	99.721502
system call	97.319483	2.680517	0.335065	99.664935
time_deviation	93.878687	6.121313	0.765164	99.234836

After analysis of the table 4.1 and 4.2 we selected DNN as the model as it has the best results for all features in table 4.1.

Table 4.3: DNN models values on merged features

	ACCURACY	**TPR**	**FNR**	**FPR**	**TNR**
dll+filename	98.98%	98.684908	1.315092	0.164387	99.835613
dll+sucess rate	99.35%	99.20841	0.79159	0.098949	99.901051
dl+ system call	98.98%	98.58564	1.41436	0.176795	99.823205
dll+time deviation	99.44%	99.264245	0.735755	0.091969	99.908031
filename+success rate	96.83%	95.758871	4.241129	0.530141	99.469859
filename+system call	**98.51%**	**97.895623**	**2.104377**	**0.263047**	**99.736953**
filename+ time deviation	96.46%	95.451717	4.548283	0.568535	99.431465
success rate+system_call	99.07%	98.753509	1.246491	0.155811	99.844189
success rate+time deviation	98.23%	97.844833	2.155167	0.269396	99.730604
system call+time deviation	98.79%	98.497047	1.502953	0.187869	99.812131

After the merging of two features we selected **filename and system call** as the provided result lies at the middle of the range of output values and can be considered as best representation for all the concatenation.

In terms of model selection, we opted for a **Deep Neural Network (DNN)** due to its impressive performance and versatility. DNNs have shown promising results in terms of accuracy and generalization across different attack and defense methodologies in cybersecurity. They exhibit strong predictive power and the ability to capture complex patterns, making them suitable for analyzing the combined dataset.

Additionally, DNNs are known for their ability to handle missing values and outliers effectively, which are common challenges in cybersecurity datasets. They can automatically learn relevant features from the data, reducing the need for manual feature engineering. DNNs can also leverage large amounts of data for training, which is often available in cybersecurity applications.

Considering the flexibility and power of DNNs in handling complex cybersecurity tasks, we chose a DNN as the preferred model for this analysis. Its ability to learn hierarchical representations and capture intricate relationships in the data makes it a valuable tool for both attacks and defense strategies in cybersecurity applications. The model's performance can be further improved by appropriately tuning its architecture, activation functions, and hyperparameters.

The DNN model is hyper-tuned and configured with one input layer, four hidden layers, and one output layer. The input layer consists of 40 nodes, as our dataset contains 40 distinct system calls. Each hidden layer contains 32 nodes, with ReLU as the activation function. The dropout rate for hidden layers is 0.2, except for the last hidden layer, the dropout rate is 0.3. Since the classification is binary, the output layer contains two nodes.The dataset is divided into 80:20 ratio where 80% belongs to the training dataset and 20% is divided equally for testing and validating the model i.e. out of 20%, 10% dataset is used for testing the dnn model whereas 10% is used to validate

the DNN model. The model is trained until 200 epochs with a batch size of 8. The model was first trained until 100 epochs, but the validation accuracy was approximately 90%. So, we increased the number of epochs to 200, and the validation accuracy dramatically increased by 8% (i.e., 98.51%).

Clustering:

After applying clustering based on family wise clustering we obtatined the following table.

Table 4.4: Number of cluster based on family

Name of the family	**Encoded value of the family**	**Number of Clusters**	**Silhouette Score Score**
adload	**0**	**3**	**0.908337904**
agent	**1**	**3**	**0.685222908**
benign	2	2	0.895408108
obfuscator	**3**	**2**	**0.925981727**
renos	4	9	0.726606942
startpage	5	8	0.732692388
vobfus	**6**	**2**	**0.703868104**
winwebsec	7	7	0.85161329
zeroaccess	**8**	**4**	**0.958644003**

Adversarial attack on Classifier

Classifier Training: Classifier is trained by the appropriate number of cluster provided by table 4.4.

Adversarial Attacks: There are common attack methods performed Fast Gradient Sign Method (FGSM), Projected Gradient Descent (PGD), Momentum iterative method(MIM) and Basic iterative method(BIM).

Transfer learning : After performing attacks, we got the result that FGSM was unable to misclassifiy our model which make it a robust model against

Table 4.5: Number of adversarial sample generated and loss accuracy

	name of family	adload		agent	obfuscator		vobfus		zeroaccess	
	cluster	1	2	1	1	2	1	2	1	2
	original no. of sample	101		64	104		91		111	
MIM	no of samples generated	no samples	5	6	2	5	3	4	17	68
	decreased accuracy		4.95%	9.38%	1.92%	4.81%	3.30%	4.40%	15.32%	61.26%
	average time per sample		26.2763	7.5022	2.3573	31.8988	3.1321	3.3509	5.1124	36.5230
PGD	no of samples generated	2		3	1	no sample	2	no sample		
	decreased accuracy	1.9801		4.69%	0.96%		2.20%			
	average time per sample	83.876		0.4353	1.5039		1.4509			
FGSM	no of samples generated	no adversarial samples								
	decreased accuracy									
	average time per sample									
BIM	no of samples generated	no samples		3	2	no samples	2	2	1	2
	decreased accuracy			4.6875	4.69%		2.20%	2.20%	0.90%	0.0180
	average time per sample			0.4353	0.5065		0.3790	0.3546	0.3049	0.3566

FGSM. Where as for MIM our model generated maximum number of samples which it most vulnerable to MIM attacks.

After getting the required adversarial samples we merged them with the training data and trained the model with the whole set of values to make our classifier more efficient against the AML attacks.

After implementing transfer learning we efficiently regained the accuracy of **98.81%** and 98.44%(TPR), 1.56% (FNR), 0.19% (FPR) & 99.81% (TNR).

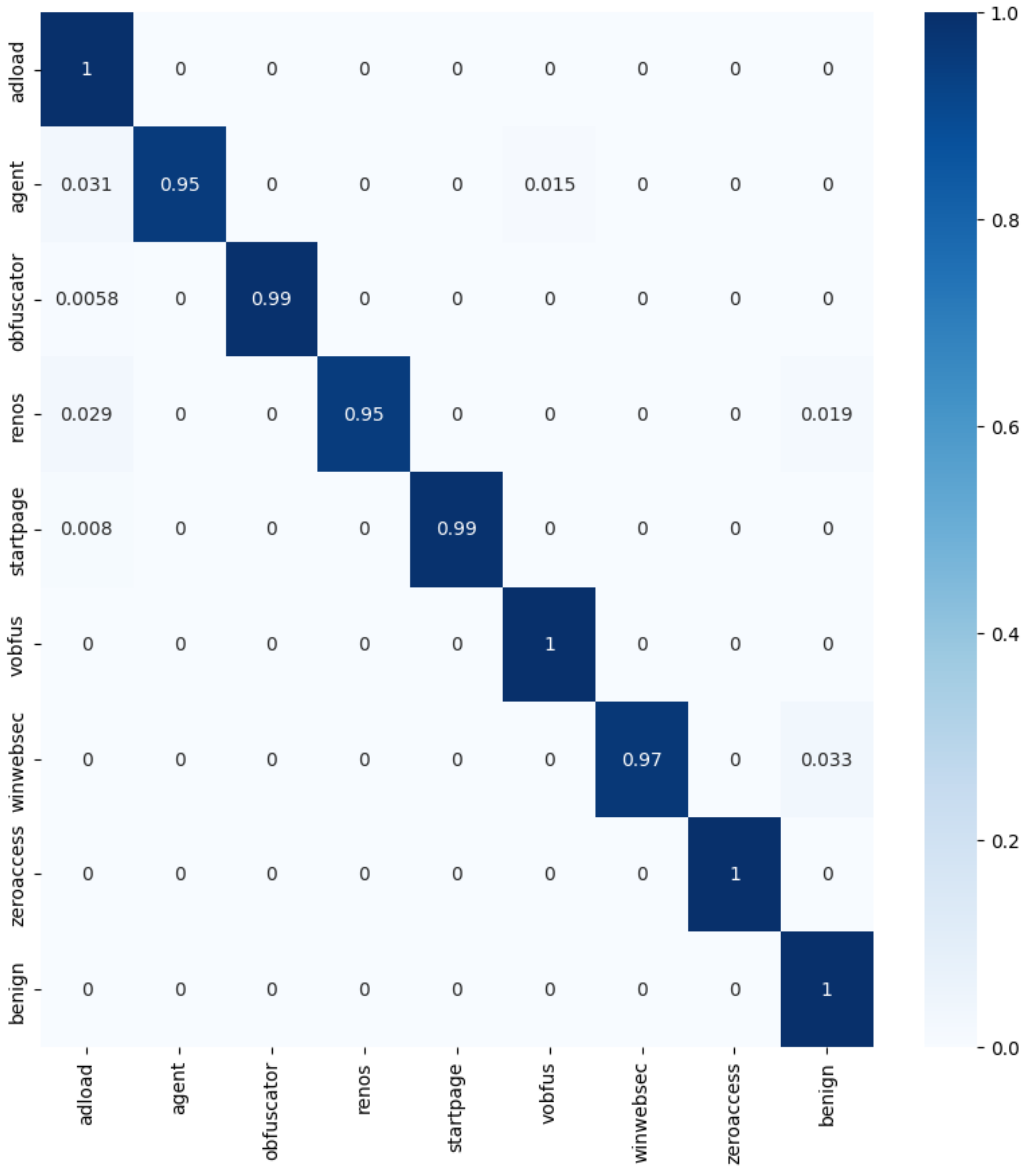

Figure 4.1: confustion matrices

Chapter 5

Conclusion

To conclude, the field of adversarial machine learning is a crucial and diverse area of research that aims to understand the limitations of machine learning models. Through this project, we have explored various techniques used to attack and defend against adversarial attacks.

We begun by examining different types of datasets with the help of multiple Machine Learning and Deep Learning models, prepared the model for evasion attacks. We also explored various attack strategies, including Fast gradient, projected gradient, momentum iterative and basic iterative. These attacks misclassifies the machine learning models, allowing adversaries to manipulate or deceive the model's predictions.To defend against adversarial attacks, we applied transfer learning, forcing to make our model robust against the attacks.

In summary, this adversarial machine learning project has provided insights into the challenges posed by adversarial attacks and the corresponding defense strategies. As the field continues to evolve, further research is necessary to enhance the robustness of machine learning models and ensure their secure and reliable deployment in real-world applications.

Bibliography

[1] S. Asha and P. Vinod, "Evaluation of adversarial machine learning tools for securing ai systems," *Cluster Computing*, pp. 1–20, 2022.

[2] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58, 2011.

[3] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[4] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE symposium on security and privacy (SP)*, pp. 582–597, IEEE, 2016.

[5] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[6] A. Dunmore, J. Jang-Jaccard, F. Sabrian, and J. Kwak, "Generative

adversarial networks for malware detection: a survey," *arXiv preprint arXiv:2302.08558*, 2023.

[7] H. Chen and F. Koushanfar, "Tutorial: Toward robust deep learning against poisoning attacks," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 3, pp. 1–15, 2023.

[8] X. Ling, L. Wu, J. Zhang, Z. Qu, W. Deng, X. Chen, Y. Qian, C. Wu, S. Ji, T. Luo, *et al.*, "Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art," *Computers & Security*, p. 103134, 2023.

[9] J. Chen, C. Yuan, J. Li, D. Tian, R. Ma, and X. Jia, "Elamd: An ensemble learning framework for adversarial malware defense," *Journal of Information Security and Applications*, vol. 75, p. 103508, 2023.

[10] R. Singhal, M. Soni, S. Bhatt, M. Khorasiya, and D. C. Jinwala, "Enhancing robustness of malware detection model against white box adversarial attacks," in *Distributed Computing and Intelligent Technology: 19th International Conference, ICDCIT 2023, Bhubaneswar, India, January 18–22, 2023, Proceedings*, pp. 181–196, Springer, 2023.

[11] S. Wu, J. Xue, Y. Wang, and Z. Kong, "Black-box evasion attack method based on confidence score of benign samples," *Electronics*, vol. 12, no. 11, p. 2346, 2023.

[12] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, *et al.*, "Technical report on the cleverhans v2. 1.0 adversarial examples library," *arXiv preprint arXiv:1610.00768*, 2016.

[13] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, *et al.*, "Adversarial robustness toolbox v1. 0.0," *arXiv preprint arXiv:1807.01069*, 2018.

[14] J. Rauber, W. Brendel, and M. Bethge, "Foolbox: A python toolbox to benchmark the robustness of machine learning models," *arXiv preprint arXiv:1707.04131*, 2017.

[15] D. Goodman, H. Xin, W. Yang, W. Yuesheng, X. Junfeng, and Z. Huan, "Advbox: a toolbox to generate adversarial examples that fool neural networks," *arXiv preprint arXiv:2001.05574*, 2020.

[16] X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang, "Deepsec: A uniform platform for security analysis of deep learning model," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 673–690, IEEE, 2019.

[17] G. W. Ding, L. Wang, and X. Jin, "Advertorch v0. 1: An adversarial robustness toolbox based on pytorch," *arXiv preprint arXiv:1902.07623*, 2019.

[18] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel, "On the (statistical) detection of adversarial examples," *arXiv preprint arXiv:1702.06280*, 2017.

[19] A. Gaur, A. Singh, A. Nautiyal, G. Kothari, P. Mishra, and A. Jha, "Deephyperv: A deep neural network based virtual memory analysis for malware detection at hypervisor-layer," in *2023 International Conference on Advances in Intelligent Computing and Applications (AICAPS)*, pp. 1–6, IEEE, 2023.

[20] Y. Zhang, Z. Liu, and Y. Jiang, "The classification and detection of malware using soft relevance evaluation," *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 309–320, 2022.

[21] L. Zheng and J. Zhang, "A new malware detection method based on vmcadr in cloud environments," *Security and Communication Networks*, vol. 2022, 2022.

[22] W. Qiang, L. Yang, and H. Jin, "Efficient and robust malware detection based on control flow traces using deep neural networks," *Computers & Security*, p. 102871, 2022.

[23] S. Shiva Darshan and C. Jaidhar, "Windows malware detection system based on lsvc recommended hybrid features," *Journal of Computer Virology and Hacking Techniques*, vol. 15, pp. 127–146, 2019.

[24] E. Amer and I. Zelinka, "A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence," *Computers & Security*, vol. 92, p. 101760, 2020.

[25] S. K. J. Rizvi, W. Aslam, M. Shahzad, S. Saleem, and M. M. Fraz, "Proud-mal: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable," *Complex & Intelligent Systems*, pp. 1–13, 2022.

[26] J. Ragaventhiran, P. Vigneshwaran, M. M. Kodabagi, S. T. Ahmed, P. Ramadoss, and P. Megantoro, "An unsupervised malware detection system for windows based system call sequences," *Malaysian Journal of Computer Science*, pp. 79–92, 2022.

[27] M. I. Yousuf, I. Anwer, A. Riasat, K. T. Zia, and S. Kim, "Windows

malware detection based on static analysis with multiple features," *PeerJ Computer Science*, vol. 9, p. e1319, 2023.

[28] A. Buriro, A. B. Buriro, T. Ahmad, S. Buriro, and S. Ullah, "Malwd&c: A quick and accurate machine learning-based approach for malware detection and categorization," *Applied Sciences*, vol. 13, no. 4, p. 2508, 2023.

[29] R. Chaganti, V. Ravi, and T. D. Pham, "A multi-view feature fusion approach for effective malware classification using deep learning," *Journal of Information Security and Applications*, vol. 72, p. 103402, 2023.

[30] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[31] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[32] A. Mądry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *stat*, vol. 1050, p. 9, 2017.

[33] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, pp. 39–57, Ieee, 2017.

[34] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin, "On evaluating adversarial robustness," *arXiv preprint arXiv:1902.06705*, 2019.

[35] S. Yan, J. Ren, W. Wang, L. Sun, W. Zhang, and Q. Yu, "A survey of adversarial attack and defense methods for malware classification in cyber security," *IEEE Communications Surveys & Tutorials*, 2022.

Appendix

Code Template 1

```
!pip install keras_tuner

import tensorflow as tf
import keras
import numpy as np
import pandas as pd
import re

import keras_tuner

import matplotlib.pyplot as plt
import seaborn as sns

from keras.models import Sequential, Model
from keras.layers import Dense
from tensorflow.keras import layers
from tensorflow.keras.layers import Dropout

from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import classification_report,
                                confusion_matrix, f1_score
```

```
# Read and divide the data
df = pd.read_csv("/content/drive/MyDrive/research/dataset2.0/
                                file_name.csv")
df=df.drop('malwares',axis=1)
```

```
df2 = pd.read_csv("/content/drive/MyDrive/research/dataset2.0/
                                    system_call.csv")
df = pd.concat([df, df2], axis = 1)
df['malwares'] = df['malwares'].apply(lambda x: re.sub(r'\d+', '
                                    ', x))
ds = df.values
X = ds[:,0:-1]
print(f"Dataset Size: {ds.shape[0]} Rows and {ds.shape[1]}
                                    Columns")
df = df.fillna(df.mean())
y=df['malwares']
```

```
encoder = LabelEncoder()
encoded_Y = encoder.fit_transform(y)
scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
X_train, x_rem, y_train, y_rem = train_test_split(X, encoded_Y,
                                    train_size=0.8, random_state=10,
                                     stratify=encoded_Y)
x_val, X_test, y_val, y_test = train_test_split(x_rem, y_rem,
                                    train_size=0.5, random_state=10,
                                     stratify=y_rem)
```

```
def call_existing_code(num_layers,units, activation,
                                    dropout_rate, lr):
    model = keras.Sequential()
    model.add(layers.Flatten(input_dim = 1675))
    for i in range(1, num_layers):
        model.add(layers.Dense(units = units, activation =
                                        activation))
        model.add(keras.layers.Dropout(rate = dropout_rate))
```

```
    model.add(Dense(20, activation= "softmax" ))
    model.compile(loss= tf.keras.losses.
                                    SparseCategoricalCrossentropy
                                    (from_logits=True) ,
                                    optimizer = "adam", metrics=
                                    [ "accuracy"])
    return model
def build_model(hp):
    num_layers = hp.Int("num_layers", 3, 8)
    units = hp.Int("units", min_value = 32, max_value=1024, step
                                     = 32)
    activation = hp.Choice("activation", ["relu"])
    dropout_rate = hp.Float("dropout_rate", 0.2, 0.5, step=0.1)
    lr = hp.Float("lr", min_value=1e-3, max_value=1e-2, sampling
                                    ="log")
    model = call_existing_code(num_layers = num_layers, units=
                                    units, activation=activation
                                    , dropout_rate =
                                    dropout_rate, lr=lr)
    return model
```

```
tuner = keras_tuner.RandomSearch(hypermodel = build_model,
                                 objective = "val_accuracy",
                                 max_trials = 5,
                                 executions_per_trial = 3)
tuner.search(X_train, y_train, epochs=3,validation_data=(x_val,
                                 y_val))
#print the summary of the search space
tuner.search_space_summary()
best_hp = tuner.get_best_hyperparameters()[0]
model = tuner.hypermodel.build(best_hp)
```

```
model.fit([X_train], y_train, epochs=150, batch_size = 8,
```

```
                                    verbose=1,  validation_data=([
                                    x_val], y_val)
# Accuracy
scores = model.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
# Processing model for metrics
predictions = model.predict([X_test])
predictions = np.argmax(predictions, axis=-1)
f1 = f1_score(y_test, predictions, average='weighted')
print('F1: ', "%.2f" % (f1*100))
```

```
# convert encoded data to labels
predictions = encoder.inverse_transform(predictions)
y_test_labels = encoder.inverse_transform(y_test)
xlab = ['adload', 'agent', 'obfuscator', 'renos', 'startpage', '
                                    vobfus', 'winwebsec', '
                                    zeroaccess', 'benign']
```

```
# Confusion Matrix
cm = confusion_matrix(y_test_labels, predictions)   # makes a
                                    confusion matrix with no of
                                    samples predicted
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]   #
                                    converts the values to
                                    percentage (cell/sum(row))
# figure size, plot heatmap, change the values to percentage
plt.subplots(figsize=(10,10))
# heatmap = sns.heatmap(cm, annot=True, xticklabels = xlab,
                                    yticklabels= xlab, vmin=0, vmax=
                                    50, fmt='.0%', cmap='Blues')
heatmap = sns.heatmap(cm, annot=True, xticklabels = xlab,
                                    yticklabels= xlab, cmap='Blues')
```

```
def get_tpr_fnr_fpr_tnr(cm):
    dict_metric = dict()
    n = len(cm[0])
    row_sums = cm.sum(axis=1)
    col_sums = cm.sum(axis=0)
    array_sum = sum(sum(cm))
    #initialize a blank nested dictionary
    for i in range(1, n+1):
        keys = str(i)
        dict_metric[keys] = {"TPR":0, "FNR":0, "FPR":0, "TNR":0}
    # calculate and store class-wise TPR, FNR, FPR, TNR
    for i in range(n):
        for j in range(n):
            if i == j:
                keys = str(i+1)
                tp = cm[i, j]
                fn = row_sums[i] - cm[i, j]
                dict_metric[keys]["TPR"] = tp / (tp + fn)
                dict_metric[keys]["FNR"] = fn / (tp + fn)
                fp = col_sums[i] - cm[i, j]
                tn = array_sum - tp - fn - fp
                dict_metric[keys]["FPR"] = fp / (fp + tn)
                dict_metric[keys]["TNR"] = tn / (fp + tn)
    return dict_metric
df = pd.DataFrame(get\_tpr\_fnr\_fpr\_tnr(cm)).transpose()
df = df.iloc[:,0:].apply(np.mean)*100
print(df)
```

Appendix

Code Template 2

```
%pip install cleverhans

import cleverhans
from keras.models import load_model

import pandas as pd
import numpy as np
import keras
import tensorflow as tf

import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics.pairwise import euclidean_distances

from cleverhans.tf2.attacks import fast_gradient_method
from cleverhans.tf2.attacks import projected_gradient_descent
from cleverhans.tf2.attacks import momentum_iterative_method
from cleverhans.tf2.attacks import basic_iterative_method

from scipy.spatial.distance import cdist

import time
```

```
file_name = "/content/drive/MyDrive/research/dnnmodel.h5"
model = load_model(file_name)
```

```
class_subsets = {}
unique_labels = np.unique(y_test)
print(unique_labels)
for label in unique_labels:
    class_subsets[label] = X_test[y_test == label]
labels = encoder.inverse_transform(y_test)
labels = np.unique(labels)
```

```
max_sil = pd.DataFrame(columns = ['Name of the family', 'Encoded
                                  value of the family','No of
                                 Clusters', 'Silhouette Score'])
for i, j in zip(unique_labels, labels):
    scores = []
    numbers = list(range(2, 10))
    print("For label ->", i, j)
    for k in numbers:
        kmeans = KMeans(n_clusters = k)
        label = kmeans.fit_predict(class_subsets[i])
        score = silhouette_score(class_subsets[i], label)
        scores.append(score)
    plt.plot(range(2, 10), scores)
    plt.title('Silhouette Score')
    plt.xlabel('Number of clusters')
    plt.ylabel('Score')
    plt.show()
    max_score = max(scores)
    max_index = scores.index(max_score)
    max_number = numbers[max_index]
    result = [j, i, max_number, max_score]
    max_sil = max_sil.append(pd.Series(result, index = max_sil.
                                       columns), ignore_index =
                                       True)
```

```
    print("************************")
max_sil.to_csv("cluster_details.csv")
```

```
clusters_per_label = []
for i, j in zip (max_sil['No of Clusters'], max_sil['Encoded
                                value of the family'] ):
    kmeans = KMeans(n_clusters = i, init='k-means++', max_iter=
                                    300, n_init=10, random_state
                                    =0)
    clusters = kmeans.fit_predict(class_subsets[j])
    clusters_per_label.append(clusters)
for label, clusters in enumerate(clusters_per_label):
    print(f"Clusters for Label {label}: {clusters}")

# For adload
cluster1_indices = np.where(clusters_per_label[0] == 0)[0]
cluster2_indices = np.where(clusters_per_label[0] == 1)[0]
cluster3_indices = np.where(clusters_per_label[0] == 2)[0]
adload_cluster1 = class_subsets[0][cluster1_indices]
adload_cluster2 = class_subsets[0][cluster2_indices]
adload_cluster3 = class_subsets[0][cluster3_indices]

# For agent
cluster1_indices = np.where(clusters_per_label[1] == 0)[0]
cluster2_indices = np.where(clusters_per_label[1] == 1)[0]
cluster3_indices = np.where(clusters_per_label[1] == 2)[0]
agent_cluster1 = class_subsets[1][cluster1_indices]
agent_cluster2 = class_subsets[1][cluster2_indices]
agent_cluster3 = class_subsets[1][cluster3_indices]

# For obfuscator
cluster1_indices = np.where(clusters_per_label[3] == 0)[0]
cluster2_indices = np.where(clusters_per_label[3] == 1)[0]
```

```
obfuscator_cluster1 = class_subsets[3][cluster1_indices]
obfuscator_cluster2 = class_subsets[3][cluster2_indices]

# For vobfus
cluster1_indices = np.where(clusters\_per_label[6] == 0)[0]
cluster2_indices = np.where(clusters_per_label[6] == 1)[0]
obfuscator_cluster1 = class_subsets[6][cluster1_indices]
obfuscator_cluster2 = class_subsets[6][cluster2_indices]

# For zeroaccess
cluster1_indices = np.where(clusters_per_label[8] == 0)[0]
cluster2_indices = np.where(clusters_per_label[8] == 1)[0]
cluster3_indices = np.where(clusters_per_label[8] == 2)[0]
cluster4_indices = np.where(clusters_per_label[8] == 3)[0]
zeroaccess_cluster1 = class_subsets[8][cluster1_indices]
zeroaccess_cluster2 = class_subsets[8][cluster2_indices]
zeroaccess_cluster3 = class_subsets[8][cluster3_indices]
zeroaccess_cluster4 = class_subsets[8][cluster4_indices]
```

```
logits_model = tf.keras.Model(model.input, model.layers[-1].
                              output)
target_label = np.reshape(2, (1,)).astype('int64')
columns_name = df.iloc[:,0:-1].columns
no_of_samples = pd.DataFrame(columns = ['No of samples in each
                              cluster', 'Techniques', 'No of
                              Adversarial samples'])
```

```
# FGSM attack for adload cluster1, folowing the same for rest of
                               clusters and family
target_class = 2
x_adv_FGSM = []
new_sample = np.array([])
df = pd.DataFrame(columns = ['Epsilon Value', 'Effort'])
```

```
df_original_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
no_adv = 0
for sample in adload_cluster1:
    flag = 0
    sample = np.reshape(sample, (1, 1675))
    start_time = time.time()
    for epsilon in np.arange(0.01, 1.0, 0.01):
        x_adv = fast_gradient_method.fast_gradient_method(
                                        logits_model, sample,
                                        epsilon, np.inf,
                                        targeted=True, y=
                                        target_label)
        predictions = model.predict(x_adv)
        predicted_class = np.argmax(predictions)
        if(predicted_class == target_class):
            end_time = time.time()
            elapsed_time = end_time - start_time
            print("Original value and  predicted value", 0,
                                            predicted_class)
            flag = 1
            no_adv = no_adv + 1
            if new_sample.size == 0:
                new_sample = x_adv
            else:
                new_sample = np.concatenate((new_sample, x_adv),
                                                axis=0)
            result = [epsilon, elapsed_time]
            df = df.append(pd.Series(result, index = df.columns)
                                        , ignore_index =
                                        True)
            df_original_adload = df_original_adload.append(pd.
```

```
                                                Series(sample, index
                                                 =
                                                df_original_adload.
                                                columns),
                ignore_index = True)
                my_array = np.reshape(x_adv, (1675,))
                df_adv_adload = df_adv_adload.append(pd.Series(
                                                my_array, index =
                                                df_adv_adload.
                                                columns),
                                                ignore_index = True)
                break
        if flag == 0:
            if new_sample.size == 0:
                new_sample = sample
            else:
                new_sample = np.concatenate((new_sample, sample),
                                                axis=0)
if(no_adv == 0):
    print("Zero adversarial sample is generated for the cluster"
                                        )
else:
    y_original = np.full((11,), 0)
    y_original = to_categorical(y_original)
    scores = model.evaluate(new_sample, y_original, verbose=0)
    adv_accuracy = scores[1]*100
    print("Accuracy after FGSM attack with  : %.2f%%" %(scores[1
                                        ]*100))
    df_adv_adload.to_csv("/content/drive/MyDrive/research/
                                        Cleverhans/FGSM/
                                        adv_samples_cluster1.csv")
    df.to_csv("/content/drive/MyDrive/research/Cleverhans/FGSM/
                                        efforts_cluster1.csv")
```

```
    df_original_adload.to_csv("/content/drive/MyDrive/research/
                                    Cleverhans/FGSM/
                                    org_samples_cluster1.csv")
values = [adload_cluster1.shape[0], 'FGSM', 'no_adv']
no_of_samples = no_of_samples.append(pd.Series(values, index =
                                no_of_samples.columns),
                                ignore_index = True)
no_of_samples.to_csv("/content/drive/MyDrive/research/Cleverhans
                                /FGSM/samples.csv")
```

```
# MIM attack for adload cluster1 and similarly applied on other
                                clusters and families
target_class = 2
x_adv_MIM = []
x_org=[]
new_sample = np.array([])
df = pd.DataFrame(columns = ['Epsilon Value', 'Effort'])
df_original_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
no_adv = 0
for sample in adload_cluster1:
    flag = 0
    sample = np.reshape(sample, (1, 1675))
    start_time = time.time()
    for epsilon in np.arange(0.01, 1.0, 0.01):
        x_adv = momentum_iterative_method.
                                        momentum_iterative_method
                                        (logits_model, sample,
                                        eps = epsilon, eps_iter
                                        = 0.009, nb_iter=100,
                                        norm= 2, y =
                                        target_label, targeted =
```

```
                                        True)
predictions = model_1.predict(x_adv)
predicted_class = np.argmax(predictions)
if(predicted_class == target_class):
    end_time = time.time()
    print("Original value and  predicted value", 0,
                                        predicted_class)
    flag = 1
    no_adv = no_adv + 1
    elapsed_time = end_time - start_time
    x_org.append(sample)
    if new_sample.size == 0:
        new_sample = x_adv
    else:
        new_sample = np.concatenate((new_sample, x_adv),
                                            axis=0)
    result = [epsilon, elapsed_time]
    df = df.append(pd.Series(result, index = df.columns)
                                        , ignore_index =
                                        True)
    df_original_adload = df_original_adload.append(pd.
                                        Series(sample, index
                                         =
                                        df_original_adload.
                                        columns),
                                        ignore_index = True)
    my_array = np.reshape(x_adv, (1675,))
    df_adv_adload = df_adv_adload.append(pd.Series(
                                        my_array, index =
                                        df_adv_adload.
                                        columns),
                                        ignore_index = True)
    break
```

```
    if flag == 0:
        if new_sample.size == 0:
            new_sample = sample
        else:
            new_sample = np.concatenate((new_sample, sample),
                                        axis=0)

if(no_adv == 0):
    print("Zero adversarial sample is generated for the cluster"
                                    )
else:
    y_original = np.full((adload_cluster1.shape[0],), 0)
    y_original = to_categorical(y_original)
    scores = model_1.evaluate(new_sample, y_original, verbose=0)
    adv_accuracy = scores[1]*100
    print("Accuracy after MIM attack with  : %.2f%%" %(scores[1]
                                    *100))
    df_adv_adload.to_csv("/content/drive/MyDrive/research/
                                    Cleverhans/MIM/
                                    adload_adv_samples_cluster1.
                                    csv")
    df.to_csv("/content/drive/MyDrive/research/Cleverhans/MIM/
                                    adload_efforts_cluster1.csv"
                                    )
    df_original_adload.to_csv
values = [adload_cluster1.shape[0], 'MIM', 'no_adv']
no_of_samples = no_of_samples.append(pd.Series(values, index =
                                no_of_samples.columns),
                                ignore_index = True)
no_of_samples.to_csv("/content/drive/MyDrive/research/Cleverhans
                                /MIM/samples.csv")
```

```
# PGD attack for adload cluster1 and similarly applied on other
```

```
                                    clusters and families
target_class = 2
new_sample = np.array([])
df = pd.DataFrame(columns = ['Epsilon Value', 'Effort'])
df_original_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
no_adv = 0
for sample in adload_cluster1:
    flag = 0
    sample = np.reshape(sample, (1, 1675))
    start_time = time.time()
    for epsilon in np.arange(0.01, 1.0, 0.01):
        x_adv = projected_gradient_descent.
                                        projected_gradient_descent
                                        (logits_model, sample,
                                        eps = epsilon, eps_iter
                                        = 0.009, nb_iter=100,
                                        norm= 2, y =
                                        target_label, targeted =
                                         True, clip_min=0,
                                        clip_max=1)
        predictions = model_1.predict(x_adv)
        predicted_class = np.argmax(predictions)
        if(predicted_class == target_class):
            print("Original value and  predicted value", 0,
                                            predicted_class)
            flag = 1
            no_adv = no_adv + 1
            end_time = time.time()
            elapsed_time = end_time - start_time
            if new_sample.size == 0:
                new_sample = x_adv
```

```
            else:
                new_sample = np.concatenate((new_sample, x_adv),
                                            axis=0)
            result = [epsilon, elapsed_time]
            df = df.append(pd.Series(result, index = df.columns)
                                        , ignore_index =
                                        True)
            my_array = np.reshape(sample, (1675,))
            df_original_adload = df_original_adload.append(pd.
                                        Series(my_array,
                                        index =
                                        df_original_adload.
                                        columns),
                                        ignore_index = True)
            my_array = np.reshape(x_adv, (1675,))
            df_adv_adload = df_adv_adload.append(pd.Series(
                                        my_array, index =
                                        df_adv_adload.
                                        columns),
                                        ignore_index = True)
            break
    if flag == 0:
        if new_sample.size == 0:
            new_sample = sample
        else:
            new_sample = np.concatenate((new_sample, sample),
                                        axis=0)

if(no_adv == 0):
    print("Zero adversarial sample is generated for the cluster"
                                    )
else:
    y_original = np.full((adload_cluster1.shape[0],), 0)
```

```
    y_original = to_categorical(y_original)
    scores = model_1.evaluate(new_sample, y_original, verbose=0)
    adv_accuracy = scores[1]*100
    print("Accuracy after PGD attack with  : %.2f%%" %(scores[1]
                                        *100))
    df_adv_adload.to_csv("/content/drive/MyDrive/research/
                                        Cleverhans/PGD/
                                        adv_samples_cluster1.csv")
    df.to_csv("/content/drive/MyDrive/research/Cleverhans/PGD/
                                        efforts_cluster1.csv")
    df_original_adload.to_csv("/content/drive/MyDrive/research/
                                        Cleverhans/PGD/
                                        org_samples_cluster1.csv")
values = [adload_cluster1.shape[0], 'PGD', 'no_adv']
no_of_samples = no_of_samples.append(pd.Series(values, index =
                                    no_of_samples.columns),
                                    ignore_index = True)
no_of_samples.to_csv("/content/drive/MyDrive/research/Cleverhans
                                    /PGD/samples.csv")
```

```
# BIM attack for adload cluster1 and similarly applied on other
                                    clusters and families
target_class = 2
new_sample = np.array([])
df = pd.DataFrame(columns = ['Epsilon Value', 'Effort'])
df_original_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
df_adv_adload = pd.DataFrame(columns = columns_name)
no_adv = 0
for sample in adload_cluster1:
    flag = 0
    sample = np.reshape(sample, (1, 1675))
    start_time = time.time()
```

```
for epsilon in np.arange(0.1, 1.0, 0.01):
    x_adv = basic_iterative_method.basic_iterative_method(
                                    logits_model, sample,
                                    eps = epsilon, eps_iter
                                    = 0.006, nb_iter=100,
                                    norm= 2, y =
                                    target_label, targeted =
                                     True)
    end_time = time.time()
    elapsed_time = end_time - start_time
    predictions = model_1.predict(x_adv)
    predicted_class = np.argmax(predictions)
    if(predicted_class == target_class):
        print("Original value and  predicted value", 0,
                                        predicted_class)
        flag = 1
        no_adv = no_adv + 1
        elapsed_time = end_time - start_time
        if new_sample.size == 0:
            new_sample = x_adv
        else:
            new_sample = np.concatenate((new_sample, x_adv),
                                            axis=0)
        result = [epsilon, elapsed_time]
        df = df.append(pd.Series(result, index = df.columns)
                                        , ignore_index =
                                        True)
        my_array = np.reshape(x_adv, (1675,))
        df_original_adload = df_original_adload.append(pd.
                                        Series(my_array,
                                        index =
                                        df_original_adload.
                                        columns),
```

```
                                                ignore_index = True)
            my_array = np.reshape(x_adv, (1675,))
            df_adv_adload = df_adv_adload.append(pd.Series(
                                                my_array, index =
                                                df_adv_adload.
                                                columns),
                                                ignore_index = True)
            break
    if flag == 0:
        if new_sample.size == 0:
            new_sample = sample
        else:
            new_sample = np.concatenate((new_sample, sample),
                                                axis=0)

if(no_adv == 0):
    print("Zero adversarial sample is generated for the cluster"
                                        )
else:
    y_original = np.full((adload_cluster1.shape[0],), 0)
    y_original = to_categorical(y_original)
    scores = model_1.evaluate(new_sample, y_original, verbose=0)
    adv_accuracy = scores[1]*100
    print("Accuracy after BIM attack with  : %.2f%%" %(scores[1]
                                        *100))
    df_adv_adload.to_csv("/content/drive/MyDrive/research/
                                        Cleverhans/BIM/
                                        adv_samples_cluster1.csv")
    df.to_csv("/content/drive/MyDrive/research/Cleverhans/BIM/
                                        efforts_cluster1.csv")
    df_original_adload.to_csv("/content/drive/MyDrive/research/
                                        Cleverhans/BIM/
                                        org_samples_cluster1.csv")
```

```
values = [adload_cluster1.shape[0], 'BIM', 'no_adv']
no_of_samples = no_of_samples.append(pd.Series(values, index =
                                    no_of_samples.columns),
                                    ignore_index = True)
no_of_samples.to_csv("/content/drive/MyDrive/research/Cleverhans
                                    /BIM/samples.csv")
```

Appendix

Code Template 3

```
# concatenated all the samples to obtain per family samples, for
                                    other families same code is
                                    applied
df = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/BIM
                                    /adv_samples_cluster1.csv")

df2 = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/
                                    MIM/adv_samples_cluster2.csv")
df = pd.concat([df, df2], axis = 0)

df2 = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/
                                    PGD/adv_samples_cluster1.csv")
df = pd.concat([df, df2], axis = 0)
ds = df.values
df=df.drop('Unnamed: 0',axis=1)
print(df.head)

single_value = 'adload'
df['malwares'] = single_value

df.to_csv('/content/drive/MyDrive/research/Cleverhans/adload_adv
                                    .csv', index=False)
```

```
# Read and divide the data
df_1 = pd.read_csv("/content/drive/MyDrive/research/dataset2.0/
                                    file_name.csv")
df_1=df_1.drop('malwares',axis=1)
```

```
df2 = pd.read_csv("/content/drive/MyDrive/research/dataset2.0/
                                    system_call.csv")
df_1 = pd.concat([df_1, df2], axis = 1)
df2 = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/
                                    adload_adv.csv")
df_1 = pd.concat([df_1, df2], axis = 0)
df2 = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/
                                    agent_adv.csv")
df_1 = pd.concat([df_1, df2], axis = 0)
df2 = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/
                                    obfuscator_adv.csv")
df_1 = pd.concat([df_1, df2], axis = 0)
df2 = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/
                                    vobfus_adv.csv")
df_1 = pd.concat([df_1, df2], axis = 0)
df2 = pd.read_csv("/content/drive/MyDrive/research/Cleverhans/
                                    zeroaccess_adv.csv")
df_1 = pd.concat([df_1, df2], axis = 0)
ds = df_1.values
X = ds[:,0:-1]
print(f"Dataset Size: {ds.shape[0]} Rows and {ds.shape[1]}
                                    Columns")


df_1.to_csv('/content/drive/MyDrive/research/Cleverhans/adv_data
                                    .csv', index=False)
```

```
df_1 = df_1.fillna(df_1.mean())
y=df_1['malwares']
df_1['malwares'] = df_1['malwares'].apply(lambda x: re.sub(r'\d+
                                    ', '', x))
```

```
encoder = LabelEncoder()
encoded_Y = encoder.fit_transform(y)
```

```
scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
X_train, x_rem, y_train, y_rem = train_test_split(X, encoded_Y,
                                  train_size=0.8, random_state=10,
                                   stratify=encoded_Y)
x_val, X_test, y_val, y_test = train_test_split(x_rem, y_rem,
                                  train_size=0.5, random_state=10,
                                   stratify=y_rem)
```

```
file_name = "/content/drive/MyDrive/research/dnnmodel.h5"
model = load_model(file_name)
```

```
# Accuracy
scores = model.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
model.fit([X_train], y_train, epochs=150, batch_size = 8,
                                  verbose=1,  validation_data=([
                                  x_val], y_val)
# Accuracy
scores = model.evaluate(X_test, y_test, verbose=0)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
# Processing model for metrics
predictions = model.predict([X_test])
predictions = np.argmax(predictions, axis=-1)
f1 = f1_score(y_test, predictions, average='weighted')
print('F1: ', "%.2f" % (f1*100))
```

```
# convert encoded data to labels
predictions = encoder.inverse_transform(predictions)
y_test_labels = encoder.inverse_transform(y_test)
```

```
xlab = ['adload', 'agent', 'obfuscator', 'renos', 'startpage', '
                                    vobfus', 'winwebsec', '
                                    zeroaccess', 'benign']
```

```
# Confusion Matrix
cm = confusion_matrix(y_test_labels, predictions)   # makes a
                                    confusion matrix with no of
                                    samples predicted
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]   #
                                    converts the values to
                                    percentage (cell/sum(row))
# figure size, plot heatmap, change the values to percentage
plt.subplots(figsize=(10,10))
# heatmap = sns.heatmap(cm, annot=True, xticklabels = xlab,
                                    yticklabels= xlab, vmin=0, vmax=
                                    50, fmt='.0%', cmap='Blues')
heatmap = sns.heatmap(cm, annot=True, xticklabels = xlab,
                                    yticklabels= xlab, cmap='Blues')
```