

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Malware, or malicious software, is any program or file that is harmful to a computer user. Its main objective is to disrupt the system's normal functioning or cause damage to the system itself and its components or both. Malware comes in many different forms, but irrespective of the type, their objective is one - damage to a system. Malware attack is a very large domain of cyber security attacks. Cryptanalysts across the world has been in a long drawn out battle which has intensified in the past decade with malware. With increase in the technological capabilities of computers, there is also a distinct sharp increase in the capabilities of what malware can do and more importantly, how a malware can prevent from being detected.

There are two main types of defense against malware: malware detection and malware prevention. This work focuses on the former and remove the problem before it even arises. There are over a billion potential systems in the world that could potentially be affected by malware. Designing a technique to prevent attacks from all different types of malware can be extremely difficult as compared to detecting the causes of malware and predicting if a machine will be hit with malware.

On Checking and analyzing different types of malware that exist at this point in time, the result is a very efficient system to guard against attacks. However, this is very difficult as it requires documentation of every malware that has ever existed, not to mention the very complicated and intricate system which requires different techniques against different types of malware. This makes malware detection essentially a time-series problem, but it is made complicated by the introduction of new machines, machines that come online and offline, machines that receive patches, machines that receive new operating systems, etc.

This leaves us in a predicament - stopping malware attacks by preparing ourselves against the different types of malware is practically impossible because it has to done exhaustively. Not doing so puts the system at great danger which defeats the purpose of defending a malware attack. Also, when technology grows leaps and bounds, so does the capability of malware.

True exhaustive analysis of malware cannot be done in theory or practice. The list of malwares is constantly growing and evolving. True preparation against attacks takes a whole different approach.

This approach is assessing vulnerability of the system rather than the attacker. If the attacker is constantly evolving and learning new techniques against the system's defense, then efforts to defend against certain types of attacks are futile. Hence, the system tries to predict an attack in a more generic sense before it has even happened by assessing the system itself. This theory is mainly based on the assumption that malware is targeted. Even though there are variants, a malware always targets a vulnerability or an exploit of the system to attack. If these weak points on the system can be found out and patch them before an attack happens, a very secure and malware proof security configuration can be developed.

1.2 Problem Statement

The project aims at developing a model which accurately predicts the probability that an operating system will be hit by a malware. Its primary goal is to detect malware among many operating systems and build a model which can accurately predict the vulnerability probability of a windows system getting affected by malware.

1.3 Objective

- To create 3 models that tries to predict which of those operating systems will be hit with any type of malware.
- Predict the vulnerability of the system and the probability that the system will be affected soon.
- An Interface for users to see their system vulnerability in a friendly manner
- Analyze and compare the models and their shortcomings and advantages over the other model

1.4 Project Deliverables

- 3 different models namely : Light Gradient Boosting Method, XDeep Factorization Method and Recurrent Neural Networks were made which tried to predict which of those operating systems will be hit with any type of malware.
- These 3 different models can predict the vulnerability of the system and give a probability if a system will be affected by malware or not.
- A seamless user interface to navigate the user to use the above said trained model.
- Out of the 3 models LGBM has the advantage of being the fastest but doesn't give a clear accuracy compared to others. On the other hand RNN although gives a definite probability but it uses a lot of resources and is very slow whereas XDeepFM gives the most appropriate accuracy.

1.5 Current Scope

Malware is software that is aimed at intentionally causing a system to behave in a way that it should not. Its main objective is to disrupt the system's normal functioning or cause damage to the system itself and its components or both. Malware comes in many different forms, but irrespective of the type, their objective is one - damage to a system. Malware attack is a very large domain of cybersecurity attacks. Cryptanalysts across the world has been in a long drawn out battle which has intensified in the past decade with malware. With increase in the technological capabilities of computers, there is also a distinct sharp increase in the capabilities of what malware can do and more importantly, how a malware can prevent from being detected. This is what our project is aimed at - Malware detection.

1.6 Future Scope

Malware can be very devastating depending on the target entity. Key-loggers can gain confidential information on entire organizations which can have monetary impact of billions of dollars. Viruses like stuxnet can change the fate of an entire country by altering war. Catching a virus before it hits a system will improve efficiency and simplify code base of various anti virus deployments. The project intends to build a model for predicting malware existence in operating systems with an impeccable accuracy. Future work could comprise of combining the works of detecting malware from the system codes written by understanding

their semantics to help predict. The project could also classify the malware being detected along with predictions.

CHAPTER 2

PROJECT ORGANIZATION

2.1 Software Process Models

In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhance the evolving versions until the complete system is implemented and ready to be deployed. An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

The software process model chosen during the development of this process was the iterative development model. As student developers the requirement collection performed at the beginning stages of application development were vague and distorted. Since the requirements could not be frozen during the initial stages the team ruled out utilizing the waterfall model. Using the iterative development model, the initial requirements of the web based user interface and a working accurate backend model were iteratively developed using JQuery and python based systems.. Initial iterations included developing the front end of the application and final stages of iterative web development involved the integration of the applications backend with this web application . The bugs in the development phase were ironed out during the testing phase of the iteration, bugs that could not be solved were added as requirements for later phases. Once the application was up and running the project team began development of the machine learning models. The iterative stages for the machine learning model development included fitting the initial model to the train set to get a higher accuracy training score. This was followed by applying the model to the test data in the test phase of the iteration to predict the vulnerability of machines to get affected by malware accurately. The model was then further tested using the real time test set obtained from the operating system information.. Once similar requirements were satisfied for each of the three

models that were being developed the requirement list was emptied and the remaining bugs in the final product were removed to complete the development process.

2.2 Roles and Responsibilities

The below Table 2.1 shows the different roles that all the four team members had. It also describes the responsibilities of each team member and what the team members as individuals took ownership of. A clear outline of the contributions of each team member is highlighted in the provided table. also It includes contributions of each team member - Devika Anil, Aravind P Anil, Abishek Padaki and Aatish Kayyath.

Table 2.1: Roles and Responsibilities of team members

Name	Role	Responsibilities
Devika Anil	Product Developer/ Project team Lead	1) Project management and scheduling. 2)Ensuring smooth communication of project status with team guide 3)Development and understanding of one prediction model. 4) Ensuring that deadlines were met by strictly sticking to schedule
Aravind P Anil	Product Developer	1) Back end development of application 2)Application integration with firebase 3)Application backend integration with the machine learning models 4)Development and understanding of one prediction model. 5) Maintaining database and modules related to working with online database.
Aatish Kayyath	Product Developer	1) Application front end development 2)Testing and debugging application.

		<p>3)Development of a hassle free UI for good user experience</p> <p>4)Development and understanding of one prediction model.</p>
Abishek Padaki	Product Developer	<p>1)Performed comparative study of ML models using data visualisation</p> <p>2)Testing and debugging application</p> <p>3)Development of a hassle free UI for good user experience</p> <p>3)Application backend integration with the machine learning models</p>

CHAPTER 3

LITERATURE SURVEY

3.1 Introduction

The papers referred to in relevance to this project revealed various techniques that have been used for Malware Obfuscation and Malware itself. Furthermore, the papers also reveal various aspects of classification of different types of malware. There are also some papers referred which are related to the Windows system and how the antivirus software on Windows works.

3.2 Related Work with Citation

3.2.1 Malware and Malware Obfuscation Techniques

Stuxnet [1] is one of the most advanced and sophisticated worms ever written. It was designed by an Intelligence agency in the United States and targeted at nuclear refinement plans in Iran. They released a worm through USB to the general public which latches on to any system and gathers administrative privileges for the system using zero-day exploits. This virus uses a total of 4 zero-day exploits. Then it searches for specific Siemens Centrifuge controllers connected to the system. If it does not exist, the worm destroys itself. Once it has control of the controller, it slowly varies the centrifuge speeds over a period of time so that they are all destroyed within a year. This Malware also increase the gas pressure to form rocks within the centrifuges so that they degrade in quality – slowly but steadily. The worm is hailed as one of the most complicated codes ever written in the history of mankind. It floated on the internet for over a year without being detected by any machine or anti-virus.

Obfuscation [2] is by far the most dominant technique along with Signature methods used by malware for evasion and staying out of reach of anti-malware systems. There are various Obfuscation techniques and this paper analyses some of those methods. The simplest

obfuscation technique is the dead code technique. It inserts harmless pieces of code at random to various parts of the payload but does not alter the behaviour in any way. The code is placed at clever locations that skip anti-malware scans. Another complementary method is Register Reassignment. Registers support legacy code. By switching from current standard to older generation of registers often, the Malware becomes very hard to find. Subroutine reordering is a very simple but effective Obfuscation technique. It generates $n!$ variants where n is the number of subroutines. It randomly rearranges the subroutine while having minimal changes on the payload or the behaviour of the malware. Instruction Substitution and Code Transposition are two other methods of obfuscation. Instruction Substitution splits a single instruction into multiple instructions while having the same effect. It effectively changes code with equivalent instructions. Code transposition reorders entire sequences of codes rather than just subroutines.

Private stream searching appears to be an entirely effective method for malware to surreptitiously search and exfiltrate email by resisting malware analysis techniques [3]. Malware designed to save and return messages on a specific sensitive topic will be able to do so without revealing the topic of interest upon analysis; all that will be determined is that it scans email in general. Furthermore, as the paper's implementation demonstrates, there is nothing to prevent these techniques from being used immediately. The example of PIR-based malware illustrates the more general possibility of malware employing public key obfuscation techniques to hide its behaviour.

The traditional security systems like Intrusion Detection System/Intrusion Prevention System and Anti-Virus (AV) software are not able to detect unknown malware as they use signature-based methods. In order to solve this issue, static and dynamic malware analysis is being used along with machine learning algorithms for malware detection and classification. The main problems with these systems are that they have high false positive and false negative rate and the process of building classification model takes time (due to large feature set) which hinders the early detection of malware. Thus, the challenge is to select a relevant set of features, so that, the classification model can be built in less time with high accuracy. The proposal presents a system that addresses both the issues mentioned above. It uses an integration of both static and dynamic analysis features of malware binaries incorporated with machine learning process for detecting Zero-day malware [4]. The proposed model is tested and validated on a real-world corpus of malicious samples. The results show that the static

and dynamic features considered together provide high accuracy for distinguishing malware binaries from clean ones and the relevant feature selection process can improve the model building time without compromising the accuracy of malware detection system.

3.2.2 Malware Detection Techniques

Malware sandboxes [5] are one of the most popular methods used by anti-virus testing engineers to find out if the effects of various malware in order to develop and create sophisticated counter measures. These are virtual testing playgrounds that are built with weak protective measures in order to get affected by a virus. However modern viruses have come up with a very new and innovative method against sandboxes in testing. The virus creators' aim is to make sure the virus or malware is not caught at the testing phase and will go under the radar of the sandbox. Hence, without a countermeasure, it can release fatal payloads onto real systems for adverse effects. The proposed methodology to evade sandboxes are by ensuring differential behaviour in sandboxes. By detecting the environment, it releases payload conditionally. This publication provides a method for sandboxes to catch such malwares with advanced evasion techniques by introducing wear and tear to the malware. Essentially, the sandbox is made as close to a real-life system as possible – specifically an old system. This is the wear and tear being introduced. It can be event logs, recycle bin size, cache entries, network entries, registry, cookie count and so on.

An analysis on malware detection and removal techniques designed for android devices [6] shows worrisome results. Mobile Industry has grown exponentially at a very large rate in the last few years. This period of time is also marked with an increase in malware count designed for android. However, the detection rates stay the same for the most part. The analysis classifies various malware evasion techniques into three categories – Trivial, DSA or static analysis, and Undetectable. The malware is taken through one of these transformations and passed into a system with an anti-malware setup on it. The study came up with many findings. The major takeaway is that all anti-malware systems on android are vulnerable to many transformations and are not up to standard. The second finding is that android anti-malware works primarily based on code level artefacts – package names, asset names etc. However, at least 43% of malware don't use these techniques at all and hence have an easy path into the system. 90% of the malware designed did not need protection against static

analysis of bytecode. The android security system does not bother implementing this vital layer on their systems and hence it is a waste of resources to enable evasion techniques for this method.

In the light of works done in regard to semantics aware methodology[7] for malware detection, aim for such technique is to understand and detect the presence of any malicious intent in a given program. Malware is known to evolve itself through the process of polymorphism or metamorphism in order to evade detection. This procedure is closely referred to as program obfuscation. Adding slightly new behavioural changes is said to modify the malware to a level where it may not be detected. The malware detection systems used a simple pattern matching technique that identified a certain sequence of instructions that is labelled malware using regular expressions. Understanding semantics of the program instruction will help overcome deficiencies brought on by the above mentioned basic technique. This also eliminates the need to frequently update the database used by the commercial virus scanners for all updates of the malware. Here, the context of malicious behaviour is found from the instruction sequence was learnt. When a template and an instruction sequence are executed from a state where the contents of the memory are the same, then after both the executions the state of the memory is the same. In other words, the malicious behavior specified by the template is demonstrated by the instruction sequence. Our malware detection algorithm AMD works by finding, for each template node, a matching node in the program. Nodes from the template and the program match if there exists an assignment to variables from the template node expression that unifies it with the program node expression. Once two matching nodes are found, whether the def-use relationships true between template nodes also hold true in the corresponding program (two nodes are related by a def-use relationship if one node's definition of a variable reaches the other node's use of the same variable) was checked. If all the nodes in the template have matching counterparts under these conditions, the algorithm has found a program fragment that satisfies the template and produces a proof of this relationship. Experimental evaluation of this algorithm has shown ability to detect all variants of certain malware, has no false positives, and is resilient to obfuscation transformations generally used by hackers.

Next study involved understanding the workings of a malware detection system used to understand and perform behaviour based analysis [8] of malware on android systems called

the crowdroid. In this system, a lightweight client is downloaded and used on smartphones as opposed to the security tools and mechanisms used in computers which are not feasible for applying on smartphones due to the excessive resource consumption and battery depletion. Then, the remote server will be in charge of parsing data, and creating a system call vector per each interaction of the users within their applications. Thus, a dataset of behavior data will be created for every application used. The more users using our Crowdroid application, the more complete and accurate will be our system. Finally, clustering each dataset using a partitional clustering algorithm was done. This way differentiation between benign applications that demonstrate very similar system call patterns, and malicious trojan applications that, even if having the same name and identifier, have a different behavior in terms of distance between example vectors. It is quite often seen that `open()`, `read()`, `access()`, `chmod()` and `chown()` are the most used system calls by malware. A benign application could make moderate or heavy use of those system calls and thus trigger false positives. Managing the perception of loss of privacy when supporting research community with their behavior information, against the benefit of having access to up-to-date behavioral-based detected malware statistics.

Normalized Compression Distance (NCD)[9] is a tool that uses compression algorithms to cluster and classify data in a wide range of applications. The author has demonstrated that several compression algorithms, lzma, bz2, zlib, and PPMZ, apparently fail to satisfy the properties of a normal compressor, and explored the implications of this on their capabilities for classifying malware with NCD. More generally, they have shown that file size is a factor that hampers the performance of NCD with these compression algorithms. Specifically, they have found that lzma performs best on this classification task when files are large, but that bz2 performs best when files are sufficiently small. They have also found zlib to generally not be useful for this task. PPMZ, in spite of being a top performer in terms of idempotence, did not come close to the most accurate compressor in any case. Finally, introducing two simple file combination techniques that improve the performance of NCD on large files with each of these compression algorithms. The authors have explored the relationship between some of these properties and file size, demonstrated that this theoretical problem is actually a practical problem for classifying malware with large file sizes, and proposed some variants of NCD that mitigate this problem.

Malware detection systems have been driven by models learned from input features collected from real or simulated environments[10]. A potential malware sample, suspicious email is deemed malicious or non malicious based on its similarity to the learned model at runtime. However, the training of the models has been historically limited to only those features available at runtime but in this paper the author has considered an alternate learning approach that trains models using “privileged” information—features available at training time but not at runtime— to improve the accuracy and resilience of detection systems. In particular they have adapted and extended recent advances in knowledge transfer and model influence to enable the use of forensic or other data unavailable at runtime in a range of security domains. The evaluation shows that privileged information increases precision and recall over a system with no privileged information: Observations are up to 7.7% relative decrease in detection error for fast-flux bot detection, 8.6% for malware traffic detection, 7.3% for malware classification. The author introduces ITect[11], a scalable approach to malware similarity detection based on information theory. ITect targets file entropy patterns in different ways to achieve 100% precision with 90% accuracy but it could target 100% recall instead. It outperforms VirusTotal for precision and accuracy on combined Kaggle and VirusShare malware. ITect opens a new front in the arms race. Its level of abstraction makes it difficult to counter and it offers scalability advantages. The authors have demonstrated excellent precision and accuracy on a representative mixture of malware types drawn from the Kaggle malware data and VirusShare. They have demonstrated that both of its constituent detectors, EnTS and SLamm, outperform previous information theoretic similarity measures. Indeed, ITect outperforms existing AntiVirus engines (as represented in VirusTotal) for accuracy and precision. Its time complexity is bounded above by the number of files to be classified. As an automated, execution agnostic, string-based similarity metric it offers wider scalability advantages beyond its time complexity class alone – reducing human effort and reducing the need for dynamic or static analysis.

With the advent of cheap computing power available freely for the general public, malware detection has become near impossible [12]. Malware uses various techniques such as Encryption, Oligomorphic, Polymorphism, Metamorphism, Obfuscation, Fragmentation, Session splicing, Protocol Violations, Code reuse Attacks and more. The most common malware detection techniques used are Signature method, Behaviour method, and Heuristic

method. However, these are simply not enough to catch the complex multi layered evasion techniques that most malwares use nowadays. The method being used here is a neural network to catch trojans. The suspected payloads are injected as weights into the neural network. The network catches the trojan when the network receives a very specific type of data.

This next research work [13] gave us an insight into a couple of data mining and machine learning models used to understand malware and detect them prior to affecting the system. Although classification methods based on shallow learning architectures, such as Support Vector Machine (SVM), Naïve Bayes (NB), Decision Tree (DT), and Artificial Neural Network (ANN), can be used to solve the above malware detection problem, deep learning has been demonstrated to be one of the most promising architectures for its superior layerwise feature learning models and can thus achieve comparable or better performance. Typical deep learning models include stacked AutoEncoders (SAEs), Deep Belief Networks with Restricted Boltzmann Machine, Convolutional Neural Networks etc. In this paper, Exploration of a deep learning architecture with SAEs model for malware detection was conducted. The SAE model is a stack of AutoEncoders, which are used as building blocks to create a deep network. An AutoEncoder, also called AutoAssociator, is an artificial neural network used for learning efficient codings. Architecturally, the form of an AutoEncoder is with an input layer, an output layer and one or more hidden layers connecting them. The goal of an AutoEncoder is to encode a representation of the input layer into the hidden layer, which is then decoded into the output layer, yielding the same (or as close as possible) value as the input layer. To form a deep network, an SAE model is created by daisy chaining AutoEncoders together, known as stacking. To use the SAEs for malware detection, a classifier needs to be added on the top layer. In this application, the SAEs and the classifier comprise the entire deep architecture model for malware detection.

The involvement of deep learning architectures such as neural networks in the classification of malware [14] is the insight given by this next research work. Results show that deep learning indeed brings improvements in classification of malware system call traces. Combining convolutional and recurrent layers was the superior idea to achieve this improvement. This combination helps us obtain slightly better results than with simpler architectures with only feedforward or only convolutional layers. Using only LSTM recurrent

network also does not achieve accuracy as high as possible with our architecture, which can be explained with the relatively short length of the malware execution traces. Although training time for neural network ranges from three to ten hours, on test time the classification is instantaneous. Therefore the neural network approach is very good in cases where there is no common need to retrain the model. Overall, this approach exhibits better performance results when compared to previous malware classification approaches.

Malware detection based on machine learning techniques [15] is often treated as a problem specific to a particular malware family but classifying samples as malware or benign based on a single model would be far more efficient. However, such an approach is extremely challenging as extracting common features from a variety of malware families might result in a model that is too generic to be useful. The author in this paper has used four different machine learning techniques — support vector machines (SVM), χ^2 test, k -NN, and random forests using n -grams as features to evaluate the effectiveness of generic malware models. To summarize, SVM, χ^2 , k -NN and random forests all performed well for individual malware families. The accuracy for all of these techniques dropped significantly when the models were more generic. Some of these techniques did better than others in the more generic cases. The random forest specifically was the strongest classifier with an accuracy of 88% for the most generic model.

PowerShell is increasingly used by cybercriminals as part of their attacks' tool chain, mainly for downloading malicious contents and for lateral movement. Indeed, a recent comprehensive technical report by Symantec dedicated to Power-Shell's abuse by cybercriminals reported on a sharp increase in the number of malicious PowerShell samples they received and in the number of penetration tools and frameworks that use PowerShell. This highlights the urgent need of developing effective methods for detecting malicious PowerShell commands. In the proposed paper, this challenge is addressed by implementing several novel detectors of malicious Powershell commands [16] and evaluating their performance. The proposal implements both “traditional” natural language processing (NLP) based detectors and detectors based on character-level convolutional neural networks (CNNs).

The method proposed [17] is an innovative method for detecting malware which uses combined features (static and dynamic) to classify whether a portable executable file is

malicious or benign. The method employs two types of neural networks to fit distinct property of respective work pipelines. The first type of neural network used is recurrent neural network that is trained for extracting behavioral features of PE file, and the second type is convolutional neural network that is applied to classify samples. At the training stage, first the static information of a PE file is extracted and sandbox is used to record system API call sequences as dynamic behaviors. Then extraction of static features based on predefined rules and dynamic features out of the trained RNN model. Next they are combined and use well design algorithm to create images. Lastly, the concurrent classifier is trained and validated using images created in the previous steps labeled with 1(malicious) or 0(benign).

3.3 Literature Survey Table of Comparison

The below Table 3.1 is a comprehensive report of every publication and the inferences drawn from it. Following the inference from the table, there is also scope for future improvements in many of the papers.

Table 3.1: Literature survey comparison

Serial No	Title and Author	Techniques Used	Inference	Future Scope
1.	Spotless Sandboxes: Evading Malware Analysis Systems using Wear-and-Tear Artefacts - Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, Michalis	Wear and tear within a sandbox to emulate a used system and game malware evasion.	Malware evasion techniques have been successfully fooled because of wear and tear but the solution is short lived as a few extra checks by the malware bypasses wear and tear	Emulation of sandbox in android systems. Extraction of real life systems but also preserve privacy.

	Polychronakis - 2017			
2.	Stuxnet - Marie Baezner, Patrice Robin - 2018	4 Zero day exploits with heavily targeted trojans	Stuxnet targeted Iranian Uranium centrifuges and was hailed as one of the most successful viruses in the world	Stuxnet was developed by the CIA and one of the most sophisticated viruses. Improvements on Stuxnet is near impossible.
3.	Catch Me If You Can: Evaluating Android Anti-Malware Against Transformation Attacks - Vaibhav Rastogi, Yan Chen, and Xuxian Jiang - 2013	Systematic tabling system to classify categorize type of android malware via transformation	Results showed android anti malware measures are very poor and simple signature modifications get through the safety network	Future improvements can be bringing the standards of android protection deployments on the same standard as desktop systems
4.	Malware Detection and Evasion with Machine Learning Techniques: A Survey - Jhonattan J Barriga, Sang	Analyzing signatures, heuristic analysis and neural network specifically	Machine learning does help with combating malware but not a single method is easily worked	

	Guun Yoo - 2017	trained to catch trojans	around. Hence hybrid techniques are needed.	
5.	Detecting Malicious PowerShell Commands using Deep Neural Networks -Danny Hendler, Shay Kels and Amir Rubin - 2018	The proposal implements both “traditional” natural language processing (NLP) based detectors and detectors based on character-level convolutional neural networks (CNNs).	PowerShell is increasingly used by cybercriminals as part of their attacks’ tool chain, mainly for downloading malicious contents and for lateral movement.	A recent technical report by Symantec reported a sharp increase in the number of malicious PowerShell samples they received and in the number of penetration tools and frameworks that use PowerShell. This highlights the urgent need of developing effective methods for detecting malicious PowerShell commands.
6.	Analysis-Resistant Malware -	Malware designed to	Private stream searching appears	The example of PIR-based

	Bethencourt, John, Dawn Xiaodong Song and Brent Waters - 2008	save and return messages on a specific sensitive topic will be able to do so without revealing the topic of interest upon analysis; all that will be determined is that it scans email in general.	to be an entirely effective method for malware to surreptitiously search and exfiltrate email by resisting malware analysis techniques.	malware illustrates the more general possibility of malware employing public key obfuscation techniques to hide its behaviour.
7.	Zero-Day Malware Detection - Ekta Gandotra, Divya Bansal, Sanjeev Sofat - 2016	It uses an integration of both static and dynamic analysis features of malware binaries incorporated with machine learning process for detecting Zero-day malware. The proposed model is tested and	The traditional security systems like Intrusion Detection System/Intrusion Prevention System and Anti-Virus software are not able to detect unknown malware as they use signature-based methods. To solve this issue, static and dynamic malware analysis is being used	The results show that the static and dynamic features considered together provide high accuracy for distinguishing malware binaries from clean ones and the relevant feature selection process can

		validated on a real-world corpus of malicious samples.	along with machine learning algorithms for malware detection and classification.	improve the model building time without compromising the accuracy of malware detection system.
8.	Semantics-aware malware detection Christodorescu, Mihai, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. 2005 IEEE Symposium on Security and Privacy	This proposal talks about understanding the semantics of the program instruction will help overcome deficiencies brought on by the above mentioned basic technique.	When a template and an instruction sequence are executed from a state where the contents of the memory are the same, then after both the executions the state of the memory is the same. In other words, the malicious behavior specified by the template is demonstrated by the instruction sequence.	Their malware detection algorithm AMD works by finding, for each template node, a matching node in the program. Nodes from the template and the program match if there exists an assignment to variables from the template node expression that unifies it with the program node

				expression.
9.	<p>DL4MD: A deep learning framework for intelligent malware detection</p> <p>Hardy, William, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li</p> <p>In Proceedings of the International Conference on Data Mining (DMIN), p. 61. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.</p>	<p>This next research work [13] gave us an insight into a couple of data mining and machine learning models used to understand malware and detect them prior to affecting the system.</p>	<p>In this paper, they explore a deep learning architecture with SAEs model for malware detection. The SAE model is a stack of AutoEncoders, which are used as building blocks to create a deep network.</p>	<p>Although classification methods based on shallow learning architectures, such as Support Vector Machine (SVM), Naïve Bayes (NB), Decision Tree (DT), and Artificial Neural Network (ANN), can be used to solve the above malware detection problem, deep learning has been demonstrated to be one of the most promising architectures for its superior layerwise feature learning</p>

				models and can thus achieve comparable or better performance.
10.	<p>Deep learning for classification of malware system call sequences.</p> <p>Kolosnjaji, Bojan, Apostolis Zarras, George Webster, and Claudia Eckert. Australasian Joint Conference on Artificial Intelligence, pp. 137-149. Springer, Cham, 2016.</p>	The involvement of deep learning architectures such as neural networks in the classification of malware.	Combining convolutional and recurrent layers was the superior idea to achieve this improvement. This combination helps us obtain slightly better results than with simpler architectures with only feedforward or only convolutional layers. Using only LSTM recurrent network also does not achieve accuracy as high as possible with our architecture, which can be explained with the relatively short length of the	Results show that deep learning indeed brings improvements in classification of malware system call traces

			malware execution traces.	
11.	<p>DL4MD: A deep learning framework for intelligent malware detection</p> <p>Hardy, William, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li</p> <p>In Proceedings of the International Conference on Data Mining (DMIN), p. 61. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.</p>	<p>This next research work [13] gave us an insight into a couple of data mining and machine learning models used to understand malware and detect them prior to affecting the system.</p>	<p>In this paper, they explore a deep learning architecture with SAEs model for malware detection. The SAE model is a stack of AutoEncoders, which are used as building blocks to create a deep network.</p>	<p>Although classification methods based on shallow learning architectures, such as Support Vector Machine (SVM), Naïve Bayes (NB), Decision Tree (DT), and Artificial Neural Network (ANN), can be used to solve the above malware detection problem, deep learning has been demonstrated to be one of the most promising architectures for its superior layerwise feature learning</p>

				models and can thus achieve comparable or better performance.
12.	<p>Deep learning for classification of malware system call sequences.</p> <p>Kolosnjaji, Bojan, Apostolis Zarras, George Webster, and Claudia Eckert. Australasian Joint Conference on Artificial Intelligence, pp. 137-149. Springer, Cham, 2016.</p>	The involvement of deep learning architectures such as neural networks in the classification of malware.	Combining convolutional and recurrent layers was the superior idea to achieve this improvement. This combination helps us obtain slightly better results than with simpler architectures with only feedforward or only convolutional layers. Using only LSTM recurrent network also does not achieve accuracy as high as possible with our architecture, which can be explained with the relatively short length of the	Results show that deep learning indeed brings improvements in classification of malware system call traces

			malware execution traces.	
13.	Detection under Privileged Information Z. Berkay Celik Pennsylvania State University Raquel Alvarez Pennsylvania State University	The evaluation shows that privileged information increases precision and recall over a system with no privileged information.	Malware detection systems have been driven by models learned from input features collected from real or simulated environments. A potential malware sample, suspicious email is deemed malicious or non malicious based on its similarity to the learned model at runtime.	The authors have adapted and extended recent advances in knowledge transfer and lpmmodel influence to enable the use of forensic or other data unavailable at runtime in a range of security domains. The evaluation shows that privileged information increases precision and recall over a system with no privileged information
14.	ITect: Scalable	The author	The authors have	ITect targets

	<p>Information Theoretic Similarity for Malware Detection</p> <p>Sukriti Bhattacharya</p>	<p>introduce ITect, a scalable approach to malware similarity detection based on information theory.</p>	<p>used a mixture of malware types drawn from the Kaggle malware data and VirusShare. They have demonstrated that both of its constituent detectors, EnTS and SLaMM, outperform previous information theoretic similarity measures.</p>	<p>file entropy patterns in different ways to achieve 100% precision with 90% accuracy but it could target 100% recall instead. It outperforms VirusTotal for precision and accuracy on combined Kaggle and VirusShare malware.</p>
--	--	--	---	---

CHAPTER 4

PROJECT MANAGEMENT PLAN

4.1 Schedule of the Project

The project plan has been prepared and proceeded as per the given requirements and the deadline given to us by the institution. The commencement of the project began this semester and has almost ended prior to our expected due date. The table 4.1 is indicative of the timeline followed by the team for the project. The timeline includes all processes involved in the project from deciding the objective to testing the final stages. The timeline also includes a gantt chart to show a better visualization of the work hours put in by the team.

Table 4.1 Schedule Plan and Timeline of the project

⊙	Name	Duration	Start	Finish	Predecessors	Resource Names
📁	Project plan	105 days	12/3/18 8:00 AM	4/26/19 5:00 PM		Devika Anil;Aatish Kayyathi;Abishek Padaki;Dr. Rajarajeshwari;Aravind P Anil
📁	Preliminary works	47 days	12/3/18 8:00 AM	2/5/19 5:00 PM		
📄	Identify topic area of research	2 days	12/3/18 8:00 AM	12/4/18 5:00 PM		
📄	Understanding current scenario	3 days	12/5/18 8:00 AM	12/7/18 5:00 PM		
📄	Finalise objectives	4 days	12/8/18 8:00 AM	12/13/18 5:00 PM		
📄	Identify technical and social relevance	2 days	12/14/18 8:00 AM	12/17/18 5:00 PM		
📄	Problem statement and synopsis	7 days	1/28/19 8:00 AM	2/5/19 5:00 PM		
📁	Review and analysis	13 days	2/6/19 8:00 AM	2/22/19 5:00 PM	2	
📄	Literature Survey	7 days	2/6/19 8:00 AM	2/14/19 5:00 PM		
📁	Feasibility Study	4 days	2/15/19 8:00 AM	2/20/19 5:00 PM		
📄	discussion of sample size and data analysis plan	2 days	2/15/19 8:00 AM	2/18/19 5:00 PM		
📄	ensure availability of research material/settings etc.	2 days	2/15/19 8:00 AM	2/18/19 5:00 PM		
📄	estimate cost and project plan	2 days	2/19/19 8:00 AM	2/20/19 5:00 PM		
📄	System design plan	2 days	2/21/19 8:00 AM	2/22/19 5:00 PM		
📄	Requirement and Scope Analysis	2 days	2/21/19 8:00 AM	2/22/19 5:00 PM		
📁	Data Analysis	17 days	2/25/19 8:00 AM	3/19/19 5:00 PM	8	
📄	Obtaining Data	3 days	2/25/19 8:00 AM	2/27/19 5:00 PM		
📄	Selection of ML models to be taken under Comparative Study	3 days	2/25/19 8:00 AM	2/27/19 5:00 PM		
📁	First Model	14 days	2/28/19 8:00 AM	3/19/19 5:00 PM	17;18	
📄	Further Literature Search	2 days	2/28/19 8:00 AM	3/1/19 5:00 PM		
📄	Draft the approach and mathematical methods	4 days	3/2/19 8:00 AM	3/7/19 5:00 PM		
📄	Implementation of code	5 days	3/8/19 8:00 AM	3/14/19 5:00 PM		
📄	Testing and Debugging	3 days	3/15/19 8:00 AM	3/19/19 5:00 PM		
📁	Second Model	14 days	2/28/19 8:00 AM	3/19/19 5:00 PM	17;18	
📄	Further Literature Search	2 days	2/28/19 8:00 AM	3/1/19 5:00 PM		
📄	Draft the approach and mathematical methods	4 days	3/2/19 8:00 AM	3/7/19 5:00 PM		
📄	Implementation of code	3 days	3/8/19 8:00 AM	3/12/19 5:00 PM		
📄	Testing and Debugging	3 days	3/15/19 8:00 AM	3/19/19 5:00 PM		
📁	Third model	14 days	2/28/19 8:00 AM	3/19/19 5:00 PM	17;18	
📄	Further Literature Search	2 days	2/28/19 8:00 AM	3/1/19 5:00 PM		
📄	Draft the approach and mathematical methods	4 days	3/2/19 8:00 AM	3/7/19 5:00 PM		
📄	Implementation of code	3 days	3/8/19 8:00 AM	3/12/19 5:00 PM		
📄	Testing and Debugging	3 days	3/15/19 8:00 AM	3/19/19 5:00 PM		
📄	Analysis and Comparative Study	3 days	3/20/19 8:00 AM	3/22/19 5:00 PM		
📄	Documentation	20 days	3/23/19 8:00 AM	4/19/19 5:00 PM		
📄	Final Report	4 days	4/20/19 8:00 AM	4/25/19 5:00 PM		
📄	Submission	1 day	4/26/19 8:00 AM	4/26/19 5:00 PM		

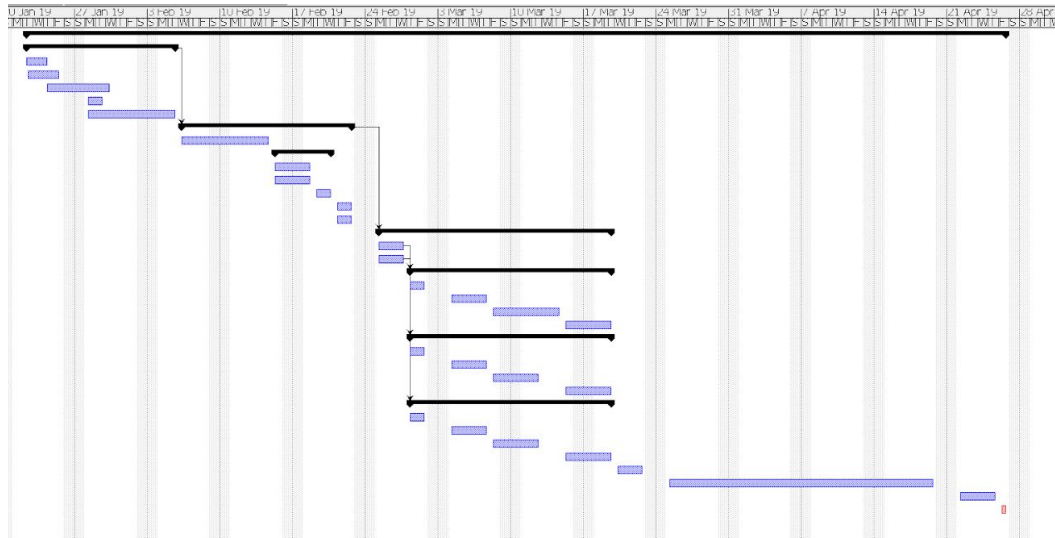


Figure 4.1 Gantt Chart illustrating the entire project schedule

As seen in figure 4.1, the gantt chart the team used describes clearly what work hours were put into the project for the required tasks. The figure also shows a clear indication of overlapping working by various team members.

A Gantt chart is a type of bar chart that illustrates a project schedule and shows the dependency relationships between activities and current schedule status. In simpler words, Gantt charts are a visual view of tasks displayed against time. They represent critical information such as who is assigned to what, duration of tasks, and overlapping activities in a project. All in all, Gantt charts are the perfect allies for planning, scheduling, and managing a project.

Some of the benefits of utilizing Gantt charts in a project are:

1. It acts as a great visualization and prioritization tool
2. Improved communication and team cohesion
3. Avoid resource overloading
4. Measure the progress of the project
5. See overlapping activities and task dependencies
6. Better time management

The sheer simplicity and ease-of-access of all relevant information make Gantt charts an ideal choice for teams to use them for organizing their schedules. Due to this, Gantt charts are widely used in project management, IT and development teams. Apart from them, marketing,

engineering, product launch, manufacturing teams can also use Gantt charts to get an overview of how things are rolling on the work front.

4.2 Risk Identification

The table 4.2 helps the team to compare analyze and mitigate any risks that may arise during the due course of the project. The table below has also been massively helpful in avoiding errors and dealing with risks early on rather than discovering them at the end stage of the project.

Table 4.2: Risk Identification and Mitigation Steps

Risk	Probability	Impact	Exp	Mitigation Plan
Failure to implement all three models and perform an in depth study of why one model is better than the other.	Medium	High	High	Identify the complexity of implementing the model to the given dataset. The process flow of the project should be broken up into small doable tasks that do not take up many work hours.
Continuous changing of requirements after models are implemented	High	Medium	Medium	Requirement Analysis should include all changes and feature bloat after the core of the project is implemented should not be entertained.
Difficulty in Integration of Deep Learning Component, Front	Medium	High	Medium	Method of Deployment and Launch into Production must be pre planned. All components can be

End And Server Deployment				written in a universal flexible language so as to easily combine and deploy them.
Lack of Proper tracking of resources and project management	High	High	High	With a team that consists of 4 developers, the team should not become sidetracked due to lack of organization. Project Management Software like Asana or Trello can be used in conjunction with tools like Kanban Boards to track progress and resources.
Failure to resolve the responsibilities to Product Designer and Developers	Low	Medium	Low	Team leader must be selected based on leadership skills Team Leader must effectively communicate to responsibilities to members and must be approachable
Insufficient QA time to validate on all browsers and OS types.	High	Low	Low	Application can be developed on the lowest version of OS as most OS support backward compatibility Appropriate time allocation in the schedule for testing and integration of the features.

Wrong time estimation (Schedule risks)	High	High	High	<p>The Process flow of the project should be broken down into small, clearly defined components where the allocated time frame for each process is relatively short in duration</p> <p>Be wary of team members or external parties, who hesitate to give estimates or whose estimates seem unrealistic based on historical data and previous experience</p>
Cost inflation of GPU Resources	Low	High	Medium	<p>GPU Estimates for training should be pre calculated and not allowed to blow up. The hardware resources needed to train large datasets must be acquired at low cost.</p> <p>AWS Spot Instances can be used to get GPU instances for cheap by bidding for certain instance types that are not heavily in demand.</p>

Security Breach of Confidential Dataset	Low	Low	Low	<p>The nature of the data being used is highly sensitive. A mixture of real and fictitious data cannot be afforded to be breached.</p> <p>All team members must take strict safety protocols to ensure the safety of the data.</p>
Front End Bloat	Low	Low	Low	<p>The front end uses relatively modern technology like JQuery for design. It has many abstraction layers which is very far from being built on the ground up.</p> <p>To avoid lag and bloat on the front end, optimization of the technology used in accordance with our project must be performed.</p>
Server Availability	Low	High	High	<p>The entire application is server-based. It runs off the flask framework in python and housed on the EC2 Instance. If the instance goes down for maintenance or due to network issues, the entire application will not run.</p> <p>Running the application off</p>

				multiple Availability Zones or AZs so it is not completely dependent on one single instance.
--	--	--	--	--

CHAPTER 5

SOFTWARE REQUIREMENT SPECIFICATIONS

5.1 Introduction

5.1.1 Purpose

Malware refers to malicious software perpetrators dispatch to infect individual computers or an entire organization's network. It exploits target system vulnerabilities, such as a bug in legitimate software (e.g., a browser or web application plugin) that can be hijacked.

A malware infiltration can be disastrous—consequences include data theft, extortion or the crippling of network systems.

In the past 5 years (since 2013), there have been more new kinds of malware created than the ten years before that combined. The need for detection of malware attacks is growing. And of particular concern are the Windows operating systems, which run on over 75% of desktops today.

There are two main types of defence against malware - malware detection and malware prevention. Focus on the former and remove the problem before it even arises. There are over a billion potential systems in the world that could potentially be affected by malware. Designing a technique to prevent attacks from all different types of malware can be extremely difficult as compared to detecting the causes of malware and predicting if a machine will be hit with malware.

5.1.2 Intended Audience and Reading Suggestions

Operating systems are a variety of system software that manages computer hardware and software resources, acting as the intermediary between programs and the hardware. In January 2019, the market share of the Windows operating system like Windows 10, Windows 7, Windows Vista, Windows XP range stood at 75.47 percent. Windows OS's for business uses and servers like Windows Server 2012, Server 2019 etc. also has a major share in the market. Because of it's huge presence in the industry Malware Detection tools are intended

for both commercial and industrial users. The tools will also be analysed by security engineers and other people related to this domain.

5.1.3 Product Scope

Over the last decade, there were lots of studies made on malware and their countermeasures. The most recent reports emphasize that the invention of malicious software is rapidly increasing. Moreover, the intensive use of networks and Internet increases the ability of the spreading and the effectiveness of this kind of software. On the other hand, researchers and manufacturers making great efforts to produce anti-malware systems with effective detection methods for better protection on computers.

Malware is software that is aimed at intentionally causing a system to behave in a way that it should not. Its main objective is to disrupt the system's normal functioning or cause damage to the system itself and its components or both. Malware comes in many different forms, but irrespective of the type, their objective is one - damage to a system. Malware attack is a very large domain of cybersecurity attacks. Cryptanalysts across the world has been in a long drawn out battle which has intensified in the past decade with malware. With increase in the technological capabilities of computers, there is also a distinct sharp increase in the capabilities of what malware can do and more importantly, how a malware can prevent from being detected. This is what our project is aimed at - Malware detection.

5.2 Overall Description

5.2.1 Product Perspective

The product is a model that's developed to accurately predict the probability that an operating system will be hit by a malware. The subsystems of the product are mainly 2 components. These subsystems are explained in the following subsections.

5.2.1.1 The Operating System

Working was done on an operating systems dataset that has over 7 million recorded operating systems and their various features. Creating a model that tries to predict which of those operating systems will be hit with any type of malware was done.

5.2.1.2 The Prediction Model

Light GBM is a very fitting model to this dataset due to the size of the data and the effectiveness and memory efficiency of this particular method of Gradient boosting. It grows in a tree wise structure. But it differs from other tree-based algorithms due to its horizontal growth structure.

Extreme Deep Factorization machine is a relatively new deep learning technique that requires no manual feature engineering. Selection of features is very important and selecting raw features often give suboptimal results. Instead using a factorization machine before applying a deep neural network for feature selection thus giving highly accurate results.

5.2.2 Product Features

1. This project aims to predict the existence of malware in a Windows system using its operating system characteristics.
2. The deep learning models using neural networks will be used to understand its accuracy in building such a model.
3. Understanding the accuracy and working of using boosting techniques like LGBM in building such a model.
4. Understanding the accuracy and working of using a hybrid like xDeepFM in building such a model.
5. Deep Neural Networks using factorization methods to understand low and high order feature interactions
6. A simple but efficient graphical user interface to give users the satisfactory analysis of their system.

5.2.3 Operating Environment

The software will run on the Windows operating system devices. All devices that support this version of the operating system will be able to run the software. The software is developed using Python based django application. The application will run on two or more virtual devices simultaneously.

The back end of the project consisting of the prediction model will be run on Jupyter notebook. This system will provide the necessary functionality to implement the data pulling and pushing services. Jupyter notebook also provides for ease of development of the classification model using Python as the scripting language.

5.2.4 Design and Implementation Constraints

1. Network or internet connections are required to interface between the phone and the real-time database.
2. Devices vary in capabilities / technology supported, and thus guarantee cannot be provided that universal access to our application across all Windows platforms.
3. Windows operating system is regularly updated by Microsoft, and future iterations may not be compatible with current version of the application.
4. The software will not be maintained by the developer following release, which may result in compatibility issues in the future.
5. The analysis is made only for Windows systems and malware detection for other operating systems like mac, or linux systems are currently not included in the product.

5.2.5 Assumptions and Dependencies

To be noted is that the current prediction model revolves around the working of Windows operating system. The data features may differ when it involves other operating systems like Linux, Mac etc.

5.3 External Interface Requirements

5.3.1 User Interfaces

The user interface is mainly run via a python GUI framework. The user will be allowed to enter the details of their operating system as input into the GUI. The GUI is also responsible for presenting a result to the user which is customized to their input. The GUI component for this project is a basic yet functional unit.

5.3.2 Hardware Interfaces

The database is very large with over 7 billion rows. This makes hardware interfaces and requirements slightly complex. Training the model directly on a user system is not a good idea as commercial laptops do not have to computational power. The computational ability is divided into two major factors - RAM and GPU. The RAM is required to load the database onto the available memory. Commercial laptops come with 8GB RAM which is below the minimum requirements to run a database of this size. Some laptops may have 16 GB RAM but even those are barely meeting the minimum requirements. When it comes to GPU, commercial systems and home systems are behind by a large margin. The best GPU available in a modern laptop is the Nvidia GTX 1060. This GPU is suited for video rendering and does not have enough CUDA cores or multiprocessing threads to deal with the raw computational power for processing and training a database. The GTX 1060 is no match for a professional processing GPU like the Tesla K80.

Both the above problems are solved by using a cloud based computational resource - which in our case is Google Colab.

5.3.3 Software Interfaces

The product will be comprised of interaction between the following software products:

5.3.3.1 Jupyter Notebook

The Jupyter notebook application allows client server interactions. It enables running of notebooks via the local browser.

5.3.3.2 Python 3.6

Python is an interpreted high-level programming language for general-purpose programming.

5.3.3.3 Google Colab

Google Colab is a free cloud service. Just like Jupyter Notebook, it enables running of notebooks via the local browser.

5.3.3.4 Windows OS

Basic operating system requirement that is required to detect possible Malware that might affect it and to interact with the above mentioned softwares.

5.4 Functional Requirements

5.4.1 Graphical User Interface Module

5.4.1.1 Description

This module allows the user to provide the details of his/her operating system via a simple interface. The model will work in the backend to analyse the given input and predict if the system will be hit by a malware.

5.4.1.2 Input / Output Sequences

The user is allowed to give all the required details of their operating system as input. The input is received via the use of forms in a python GUI framework. The GUI framework also provides an output which indicates the chance of a particular system being hit by malware.

5.4.1.3 Functional Requirements

Req 1 - Must have a working internet connection that allows him/her to connect to a network which enables connection to the trained model to predict the probability of being hit by malware.

Req 2 - A system that has the software and hardware capabilities to support the python GUI framework

5.4.2 Light GBM Model

5.4.2.1 Description

Light GBM is a very fitting model to this dataset due to the size of the data and the effectiveness and memory efficiency of this particular method of Gradient boosting. It grows in a tree wise structure. But it differs from other tree-based algorithms due to its horizontal growth structure.

5.4.2.2 Input / Output Sequences

The input to the LGBM model is the training data set which is used to calculate and predict the probability of a malware infection. There is also a test dataset of smaller size which is derived from the user.

5.4.2.3 Functional Requirements

Req 1 - The system must have access to the test dataset to provide accurate output to the user. The system must have full connectivity to the user data to produce an output.

5.4.3 XDeepFM Model

5.4.3.1 Description

Extreme Deep Factorization machine is a relatively new deep learning technique that requires no manual feature engineering. Selection of features is very important and selecting raw features often give suboptimal results. Instead a factorization machine before applying a deep neural network for feature selection thus giving highly accurate results was used.

5.4.3.2 Input / Output Sequences

The input to the XDeepFM model is the training data set which is used to calculate and predict the probability of a malware infection. There is also a test dataset of smaller size which is derived from the user.

5.4.3.3 Functional Requirements

Req 1 - The system must have access to the test dataset to provide accurate output to the user.
The system must have full connectivity to the user data to produce an output.

5.4.4 Recurrent Neural Network

5.4.4.1 Description

Recurrent Neural Network is one of the cornerstones of Deep learning. Being one of the oldest yet simplest and reliable models, it works very well for the presented data as computation power is cheaply available. Even if it is not the most efficient method in terms of power consumption, it gives us accurate results with the right hyper parameter tuning. It does have its own issues such as vanishing gradients where the gradient becomes very small. Since the network is feedforward, this reinforces in the future steps making the gradient smaller with each step until it becomes insignificant. This problem is countered using Long Short-Term Memory.

5.4.4.2 Input / Output Sequences

The input to the XDeepFM model is the training data set which is used to calculate and predict the probability of a malware infection. There is also a test dataset of smaller size which is derived from the user.

5.4.4.3 Functional Requirements

Req 1 - The system must have access to the test dataset to provide accurate output to the user.
The system must have full connectivity to the user data to produce an output.

5.4.5 Data Conversion

5.4.5.1 Description

The GUI component for python will take data in JSON form which is provided by the testing data. This data however cannot be used in raw form and must be converted to a usable state.

Hence this module converts the data from JSON to CSV so that it can be consumed by the GUI component to perform the operations needed on the training data.

5.4.5.2 Input / Output Sequences

Input will be the JSON format data acquired from the database which can be extracted as training data. The output is CSV formatted data that can be used by the GUI and supplied to the trained model behind the GUI.

5.4.5.3 Functional Requirements

Req 1 - The system must have access to the test dataset to provide accurate output to the user. The system must have full connectivity to the user data to produce an output.

Req 2 - A conversion framework within python which enables the data conversion is required.

5.5 Non-Functional Requirements

5.5.1 Safety Requirements

The details about any form of user authentication used in the graphical user interface are safely stored and will not be used for any other purpose than for the authentication system in the current application. The same applies for the data information obtained regarding the operating system state. It will be used solely for the malware prediction purpose as well as the notification of the presence of such malware. Also the data is not just stored on the machine, and so it can be retrieved on any notion of system crashing.

5.5.2 Security Requirements

The graphical user interface, running as an application on the Windows device should need no additional information other than collected operating system parameters. The rest should be taken care by the wireless security settings on the device which must allow for the application to connect to the server so it can feed information to and from the server. Otherwise, access to the user's personal information from other apps, i.e. calendar information, email, contacts, photos, other web applications etc. is under no circumstance

necessary and should be considered a breach of privacy in the event of it being used for any other purpose.

5.5.3 Software Quality Attributes

If the internet service gets disrupted while sending information to the analytics module, the information can be sent again for verification. The software should run smoothly without crashing or freezing, regardless of any machine learning parameters. It should have a very intuitive interface that is easy to learn so the user can focus mostly on their work other than the scanning process. Optimally, the architecture of the application might also be flexible enough to be easily adapted for a device with other operating systems or even android and ios devices. At the end of the project, all source code, documentation, as well as any other material related to the development of the application may be made freely available to other developers, where it may be used as a reference or for further development.

5.5.4 Business Rules

Any individual may have use of this project for academic or personal use. The project is part of the development of the open-source project; the code, documents, or other materials used for this project cannot be used for commercial purposes, without the required permissions from the developer's side. Others wishing to further develop the code after the project's completion are free to do so, with the required permissions from the developer's side.

5.5.5 Performance Requirements

The objective of this project is to develop a reliable multi-headed model for predicting the vulnerability of getting affected by malware in a Windows system. Also, the analysis is not performed directly on the graphical user interface but using the Flask platform to extract the data from the GUI for analysis by a model that will already be trained. This helps in loading of the application very quickly i.e. the response time is very less. Another aspect of performance requirement included are the mechanisms that will be included to ensure appropriate messages are prompted when the server is unable to process requests. Also, the efficiency of the Machine Learning model used for clarification is the one that has performed the highest in the comparative study in the analysis portion of the project.

CHAPTER 6

DESIGN

6.1 Introduction

Malware, or “malicious software,” is an umbrella term that describes any malicious program or code that is harmful to systems. Hostile, intrusive, and intentionally nasty, malware seeks to invade, damage, or disable computers, computer systems, networks, tablets, and mobile devices, often by taking partial control over a device’s operations. Malware comes in many different forms, but irrespective of the type, their objective is one - damage to a system.

Malware attack is a very large domain of cybersecurity attacks. Cryptanalysts across the world has been in a long drawn out battle which has intensified in the past decade with malware. With increase in the technological capabilities of computers, there is also a distinct sharp increase in the capabilities of what malware can do and more importantly, how a malware can prevent from being detected. So objective of this project - rather than inspecting malware, much better results can be obtained by analysing the system and its own vulnerabilities against targeted attacks from malware. The outcome is a predictive model that outlines the probability of a system with many parameters being affected in the near future or not. A generic solution that fits all the systems will help plug holes in many security systems - primarily windows machines as they are the most used operating system on the planet currently. This is what our project is aimed at - Malware detection.

6.1.1 Purpose

The purpose of System Design document is to translate the requirements and the elements involved in designing the processes into a technical design that will be used to develop the application.

6.2 Architecture Design

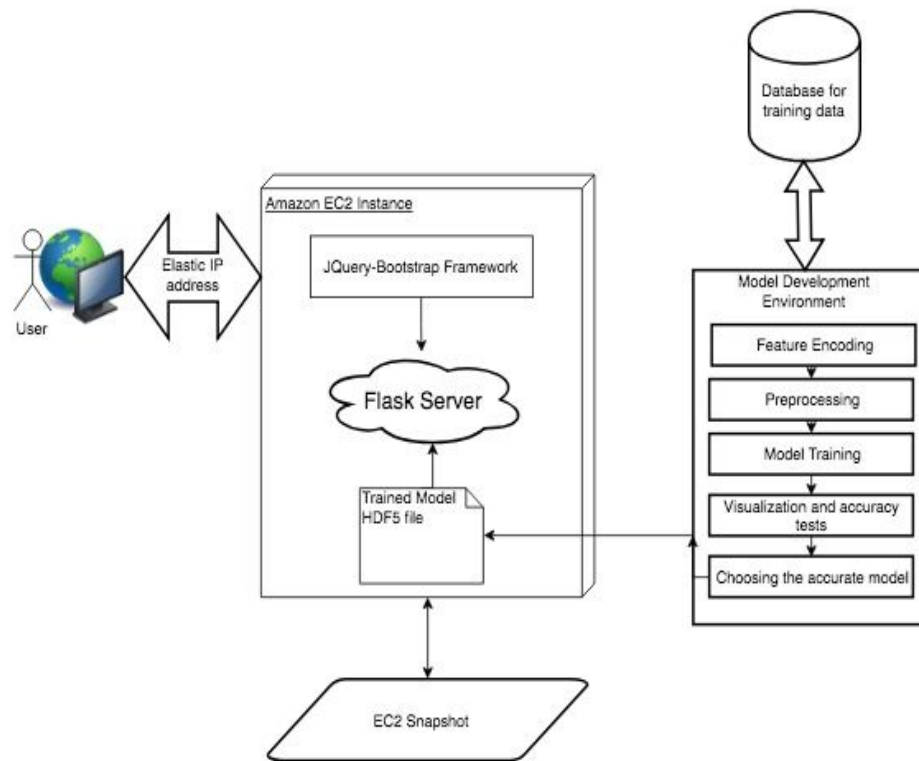


Figure 6.1: High-Level Architecture Design of EC2 Module

Figure 6.1 shows us the architecture and a birds eye view of the entire project. It involves all interactions between the modules of the project as well as high level module to module communication.

The final product has to be available publicly to anybody. Hence, rather than running it off a local system, the model and the GUI run on an EC2 instance with a public elastic IP address. On the instance, the GUI has been created using CSS and bootstrap along with all the assets housed on the same virtual computer. The model is connected to the frontend via a server whose routes and API endpoints are created using a flask server. The server fetches the trained model details from an h5 file which stores the model itself. Storage of a small sample of the dataset within the EC2 instance is also performed.

6.3 Detailed Software Design

6.3.1 Elastic Compute Cloud and Server Hosting

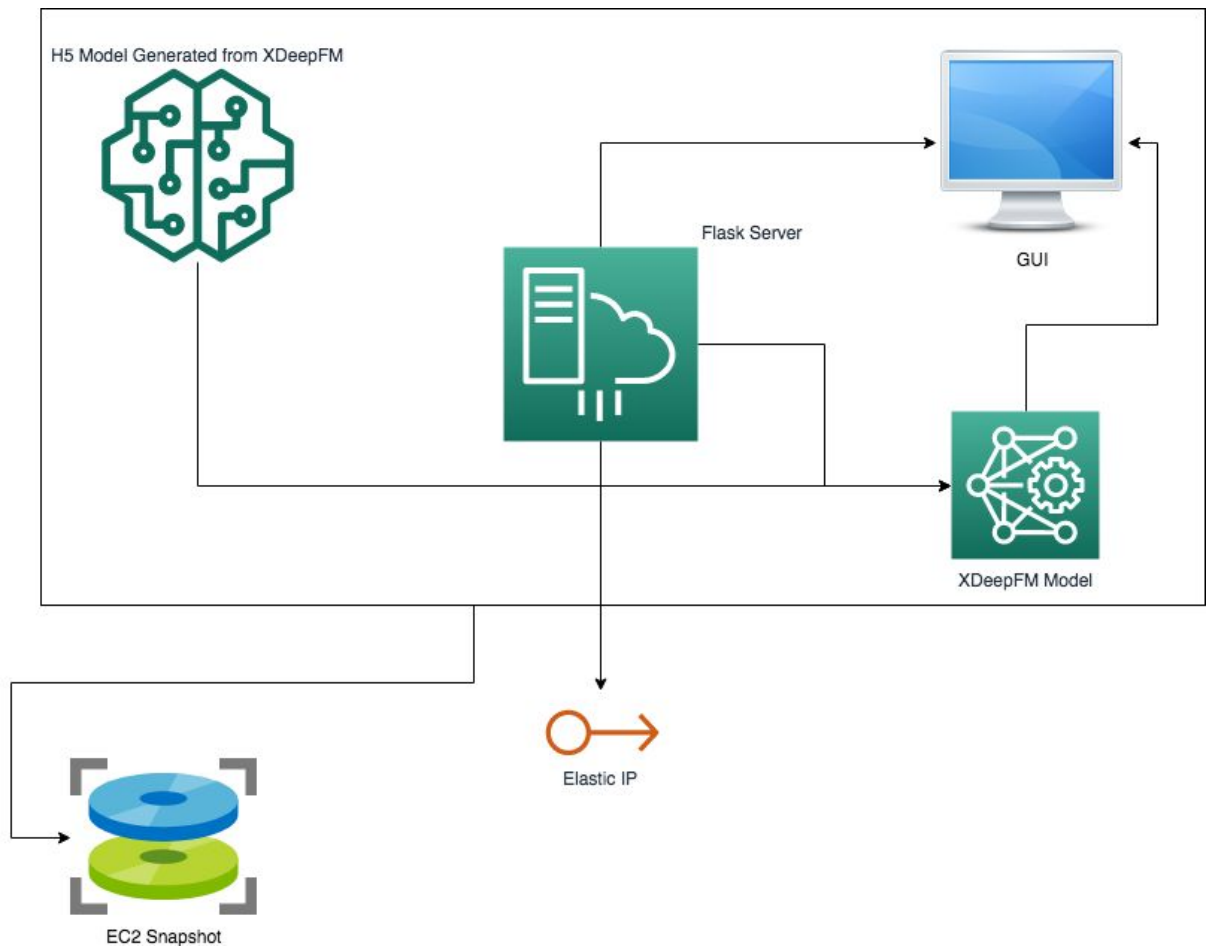


Figure 6.2 EC2 Instance Architecture of the System

Figure 6.2 illustrates the EC2 Architecture and how it houses the flask server, h5 model and the python file which outputs the model. The Flask server also renders the GUI to the public using elastic IP. The EC2 Instance is a cloud computing solution from AWS. It is an entire computing system on the cloud. The platform was launched on the cloud so the general public can view the application and not just the developers.

The core of the EC2 is the flask server. The system also contains an Elastic IP which is another resource from AWS. It gives the cloud compute system an address that the public can reach the instance. On hosting web server, then the application GUI can be accessed directly using the Elastic IP. The system uses an elastic IP over auto assigned public IP as an Elastic IP would not change in the due course of time when the instance is stopped and started.

The Flask server serves 2 components - The model and the User interface. The model works by using a pre generated and pre trained h5 file. This file represents the trained model so the system does not have to waste resources each time it runs. The python function uses this h5file to generate a vulnerability report from the new data. The system then sends the report along with model data to the GUI component which is also served by the flask server.

The entire EC2 instance is also backed up regularly using incremental backups with an AWS feature called EC2 Snapshot. It versions the instance as a whole at intervals thus ensuring loss of data is not a possibility.

6.3.2 Model Design

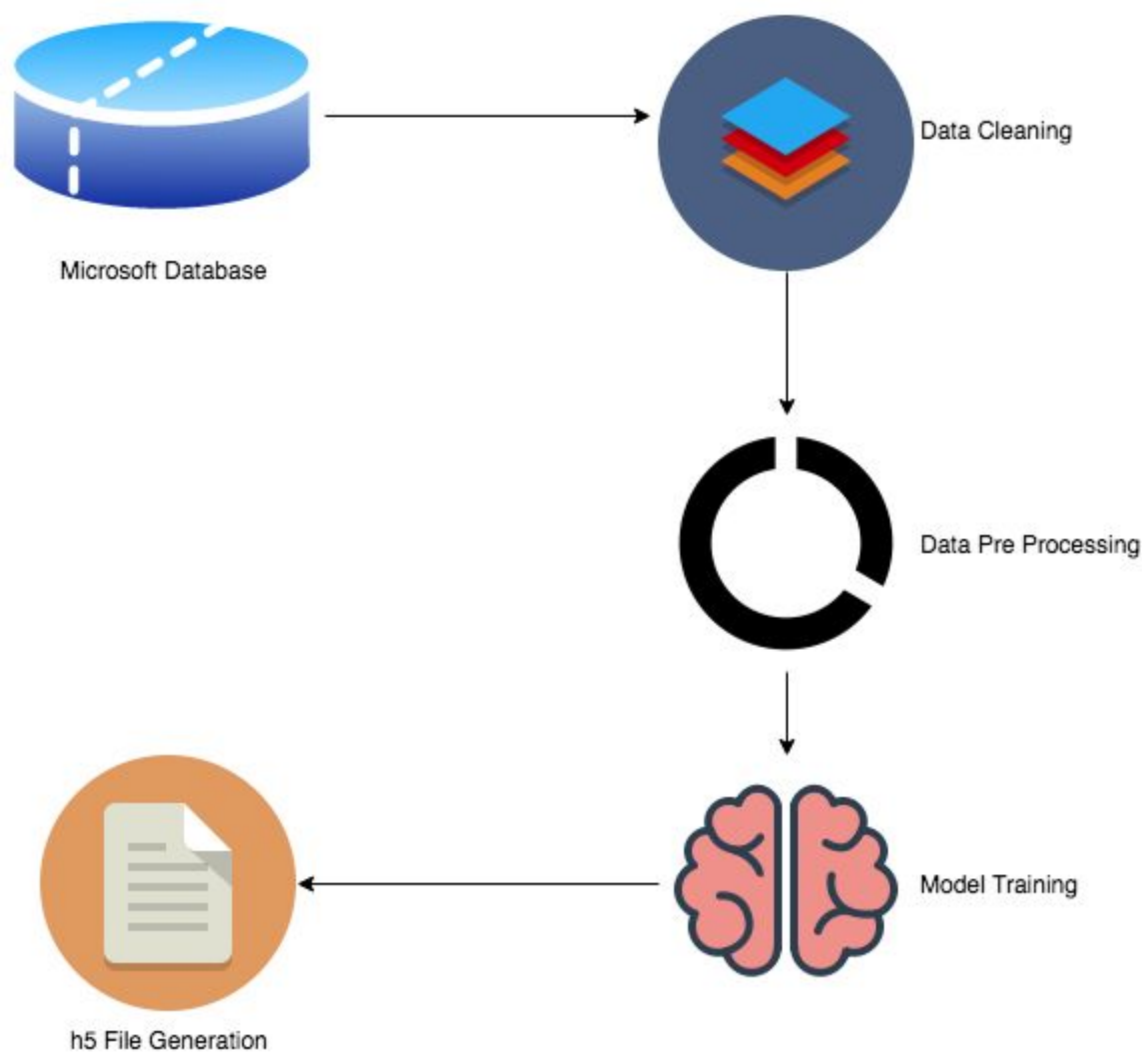


Figure 6.3 Model Development Environment Architecture of the System

Figure 6.3 shows the step by step process of h5 file generation which is uploaded to the EC2 Instance. It starts from the database and goes through various processing stages.

The development of each model is a multi part process which includes dealing with a lot of data processing. Initially it requires cleaning the original dataset and removing the unnecessary features. This saves the team a lot of time as training takes lesser resources with lesser features.

After data cleaning, data pre processing is conducted. This is a vital step as the final accuracy receives a 35% boost due to pre processing. There are two pre processing methods involved - Time Split Validation and Feature Encoding.

This is followed by training of the model. Training was done using GPU as CPU would be a very expensive hardware component for this task. Also, the CPU architecture is very unsuited and wasted for a task like deep learning on a dataset of the current nature.

Finally, the system use pre built python libraries to generate the h5 file. The h5 file is a representation of the trained model and can be thought of as a final end product to training each model. This trained model can be easily exported and operated upon given the system has access to the dataset. The h5 file is housed on the EC2 instance. The flask server contains functionality to access the h5 file. With this access, the flask server can then generate new data and get a new accuracy report on the generated data using the h5 file.

6.3.3 Use Case Diagram

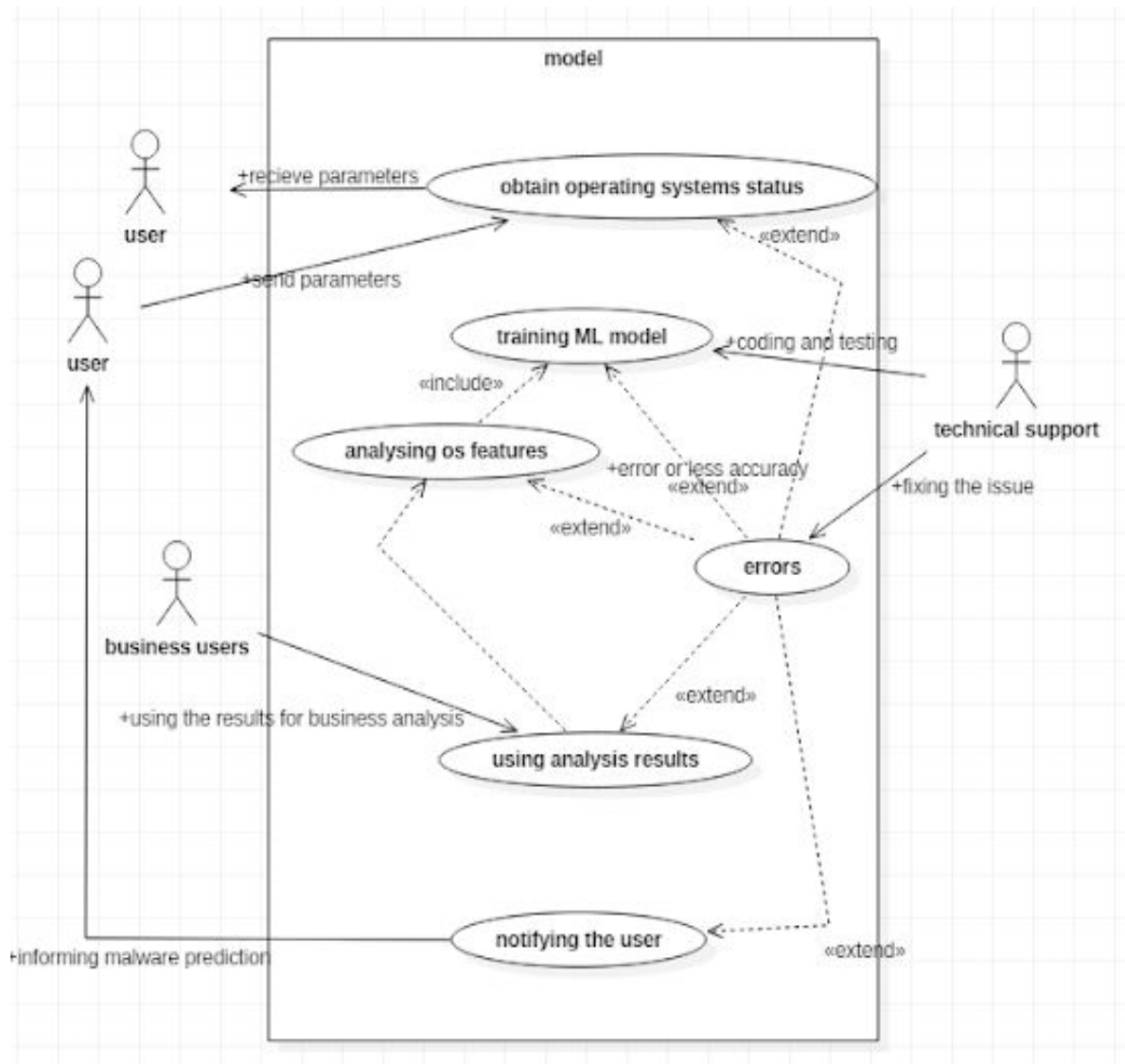


Figure 6.4: Use Case Diagram -Windows malware prediction system

Figure 6.4 shows the Use Case Diagram. It highlights the actors and processes involved. The translation of functional requirements of the project into design principles is clearly highlighted in figure.

6.3.4 Sequence Diagram

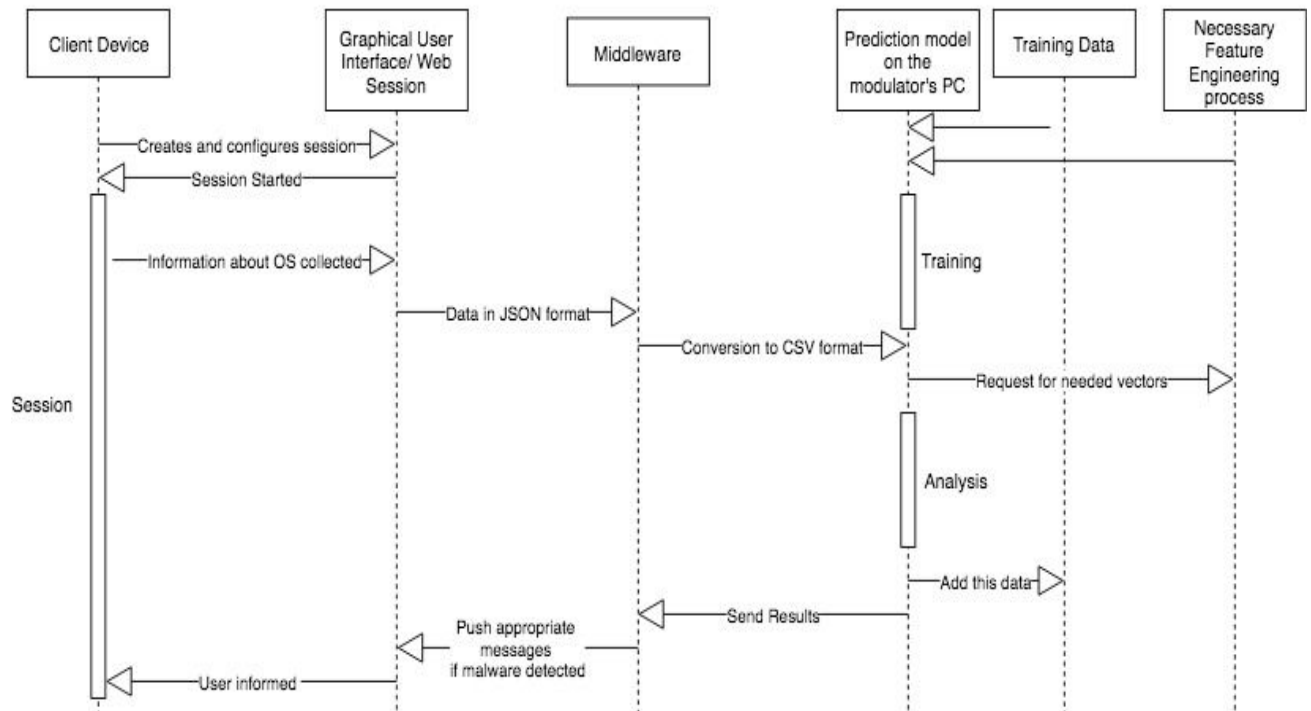


Figure 6.5: Sequence Diagram - Windows Malware Prediction System

Figure 6.5 shows high level interactions among the models and the users. The sequence diagram is representative of the order and time sensitivity of the functions represented in the UML diagram in figure 6.4.

6.4 Detailed Hardware Design

Entirety of the hardware used for operational procedures has been obtained from AWS.

6.4.1 Central Processing Unit

The t2 large uses an Intel Xeon E5 Processor which has been custom designed for AWS due to its long standing relationship with Intel. For a t2 large, 2 Intel Xeon CPUs is Haswell based and has a base speed of 2.9 GHz with turbo boost up to 3.5 GHz which is activated when the silicon chip reaches a critical temperature.

The Xeon series is different from consumer grade CPUs of the same Haswell or Kaby Lake Architecture. The server grade architecture is built for speed and resilience where a consumer grade CPU would wear out with consistent usage over a prolonged period of time.

6.4.2 Graphical Processing Unit

The system uses two different architectures of GPU for our model training. The first is the free GPU provided by Google Colab. It is based on old Kepler architecture and is a tried and tested machine for training deep neural networks.

The second GPU used is a consumer specific GPU which is the Nvidia GTX 1080. Even though the primary purpose of the 1080 is video rendering, gaming applications and mining cryptocurrency, this GPU is powerful enough to make up for its lack of ram in CUDA cores and Parallelism Speed. Even though the card may run hotter, it can train our models faster.

6.4.2.1 Tesla K80

The tesla K80 is a veteran in the neural net industry. Widely used by cloud architecture giants like Amazon, Google, PaperSpace, Azure and IBM, it is a preferred choice for neural net training despite its old architecture due to two main reasons -

- Undervolting - Since the card is undervolted and binned at the manufacturer's end, it runs safer and cooler than other cards which may be more powerful. Cooling costs are reduced and its bare metal performance makes this GPU a very easy to work with choice for everybody.
- Power Efficiency - The card runs much cheaper on the performance per power scale which makes it first running option ahead of other GPUs when it comes to scalability. Even though the cost savings are minimal with one GPU, massive applications which require over 4 GPUs can see significant cost savings due to how efficiently the K80 runs.

6.4.2.2 Nvidia GTX 1080

The GTX 1080 is a commercial GPU meant for video rendering and the gaming industry. It was also widely used for mining with multiple GPUs and combined with it's upgraded

version - the GTX 1080 Ti for mining cryptocurrency. A comparison chart between the Nvidia GTX 1080 and the Tesla K80 should highlight the major differences.

Table 6.1: GPU Comparison between Tesla K80 and GTX 1080

<i>Feature</i>	<i>Tesla K80</i>	<i>GTX 1080</i>
Memory Speed	5012 Mhz	10008 Mhz
FLOPS	4.113 TFLOPS	8228 GFLOPS
Texture Rate	171.4 GTexel/s	257.1 GTexel/s
Shading Units	2496	2560
Mapping Units	208	160
Clock Speed	562 Mhz	1607 Mhz
Price	Rs 280,000	Rs 50,000
Architecture	Kepler 2.0	Pascal
TDP	300 W	150 W

As inferred from Table 6.1, the GTX 1080 severely lags behind in Floating Operations. However, the table clearly shows that the GTX 1080 is superior in every other aspect. But due to its large memory, clock speed and low TDP, the GTX 1080 wins by a huge margin. It is much more suited to the project than the Tesla. The 1080 also runs on much newer architecture while having twice as much Memory Cap.

6.6 Data Flow Diagram

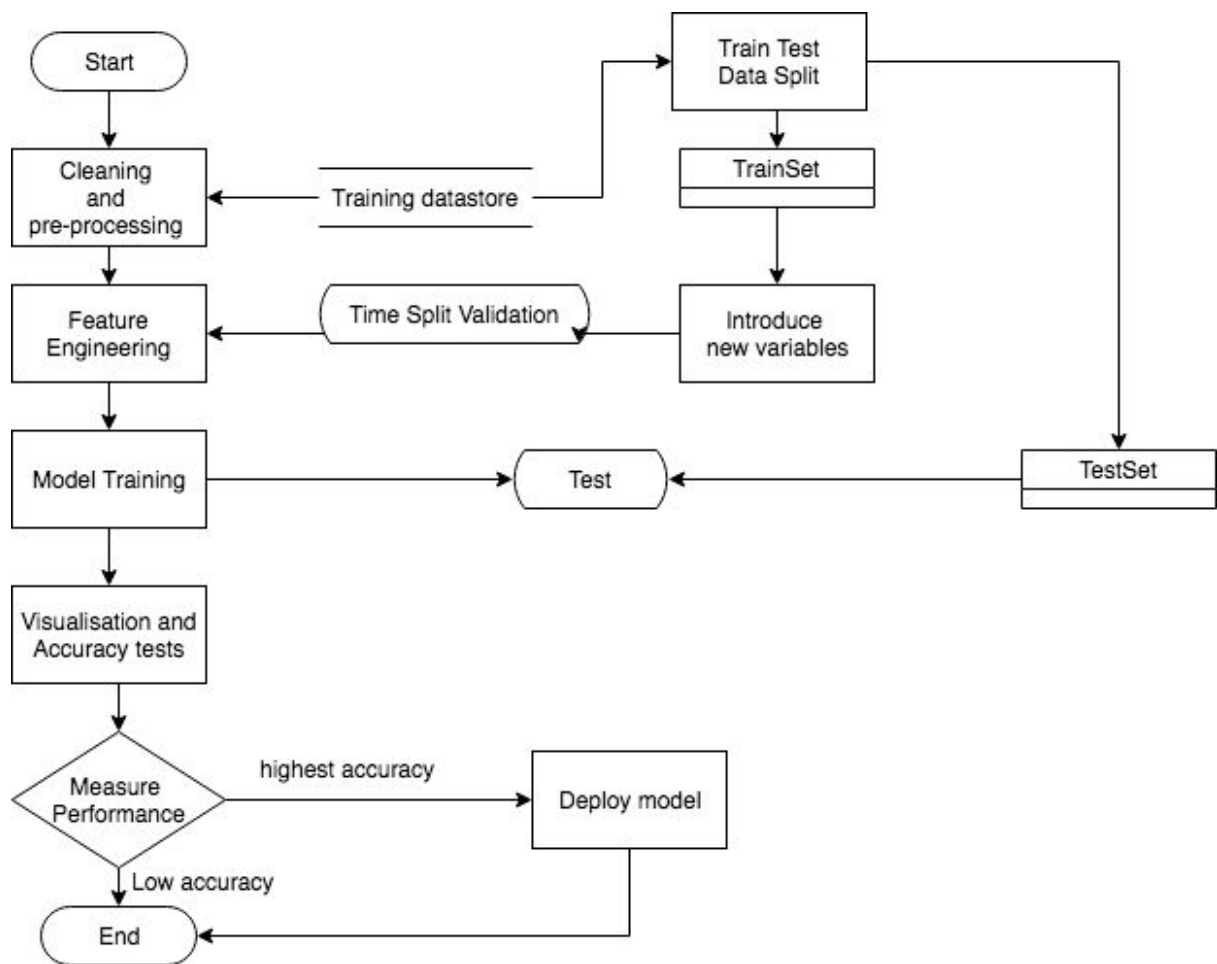


Figure 6.6: Data Flow Diagram - Windows Malware Prediction System

As we can see in figure 6.6, the data flow diagram shows the architecture pathway for movement of data. The training takes a different route compared to testing which is more elaborate.

6.7 Module Diagram

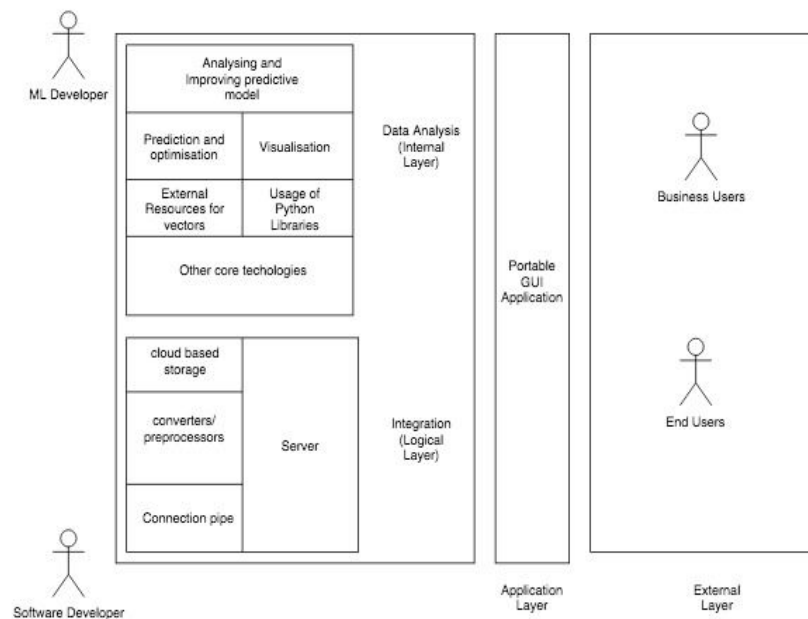


Figure 6.7: Module Diagram of models and Flask server

The module diagram shown above in figure 6.7 is illustrative of the inter and intra dependencies of all the modules involved. The figure helps the team understand the structure of existing systems and must be referred to before changing any modules of the project due to the dependencies.

6.8 System Security and Integrity Controls

The system works with a very large database which contains details of Operating Systems. Whether these are actual instances or generated by permutations and combinations of various options is unclear. However, data security is a high priority task and must be ensured. At no point is the data uploaded to an arbitrary cloud or any anonymous storage service. The only portion where data reaches the internet is during the initial training of the model where the code is run on a powerful machine so as to get faster results.

The codebase is also not stored on any cloud or virtual machine instance at any point of time. The GUI is run completely on a local machine and leak of data is considered a breach of privacy even if it seems highly improbable. The lack of clarity about the extent of sensitivity

of data should not interfere with the security of the entirety of the system as the system security should be given high priority. There are no instances or points on the timeline of the system where authorization, identification or authentication is required. Hence, the probability of a breach of data occurring is miniscule. Even if the three modules seem separate, the system works as one functional unit. The python GUI blends in with the models and presents us with an integrated working environment for smooth functioning. The database sits in the backend and does not interfere with the models. However, it still remains a vital part of the system and is linked to the frontend because the input to the GUI is eventually a part of the database.

CHAPTER 7

IMPLEMENTATION

7.1 Tools and Technology Introduction

7.1.1 Jupyter Notebook

The Jupyter notebook application allows client server interactions. It enables running of notebooks via the local browser. The use of Jupyter notebook requires no active internet connection, however since our application is interacting with a real time data hosted on firebase an active internet connection is a must. Jupyter notebooks runs as an interpreter which enables line by line code execution. This makes it very easy to detect errors in the code and correct them. This is especially useful as model construction for classification will often take several minutes to run and using an interpreter will produce faster error detection results.

7.1.2 Python 3.6

Python is an interpreted high-level programming language for general-purpose programming. Python provides a large number of libraries and a variety of inbuilt functions. Utilization of these functions is key to the development of the model the team intends to use of the classification. Furthermore, python also provides easy methods to read into csv files and JSON files both of which are file formats that will be commonly used during the course of this project.

7.1.3 Google Colab

Google Colab is a free cloud service. Just like Jupyter Notebook, it enables running of notebooks via the local browser. It runs as an interpreter which enables line by line code execution. This makes it very easy to detect errors in the code and correct them. This is especially useful as model construction for classification will often take several minutes to run and using an interpreter will produce faster error detection results. The most important feature that distinguishes Colab from other free cloud services is Colab provides GPU and is

totally free. It helps develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and OpenCV.

7.1.4 Flask Server

Flask Server is a Python based server side scripting extension which helps us in hosting our entire platform on a server. It is used in our project to create API endpoints. It uses Werkzeug tools in a WSGI platform to integrate and scale our project. It can take multiple requests at the same time. Our choice of Flask for server side scripting is because the team have our backend Deep Learning and having the middleware as well as backend on the same Language makes things easier.

Flask Server is also easy to deploy via docker or terminal. It is a better tool for our use case than Django. Django requires configuration for database on the backend which is overkill for this project. The Server is very simple to implement and can render templates with its own WSGI server.

7.1.5 AWS EC2 Instance

Elastic Cloud Compute is an AWS service that is used to launch applications into the cloud easily. EC2 is used to launch entire virtual machines on the cloud with ease and easily scalable. Amazon provides cheap cloud solutions which are priced based on the hour. Our instance is t2 which means it is a general purpose instance since not many users will send in requests to our application simultaneously.

7.2 Feature Engineering Performed

Time split validation, feature encoding, and elimination of features by variable importance and cross-validation techniques are the secret ingredients to improving accuracy that is used in all the models under this study.

7.2.1 Time Split Validation

The following variables were discovered by trying engineered variables, based on the knowledge of operating systems and malware, to see which ones increased Time Split

Validation. Each variable was added to the model one at a time and validation score was recorded. Only the following 5 variables increased validation score.

AppVersion2 indicates whether your Windows Defender is up to date. This is the second number from AppVersion. Regardless of your operating system and version, you can always have AppVersion with second number equal 18. For example, you should have 4.18.1807.18075 instead of 4.12.xx.xx.

Lag1 is the difference between AvSigVersion_Date and Census_OSVersion_Date. Since AvSigVersion is the virus definitions for Windows Defender, this variable indicates whether Windows Defender is out-of-date by comparing its last install with the date of the operating system. Out-of-date antivirus indicates that a user either has better antivirus or they don't use their computer often. In either case, they have less HasDetections.

Lag5 is the difference between AvSigVersion_Date and July 26, 2018. The first observation in Microsoft's training data is July 26, 2018. Therefore if a computer has AvSigVersion_Date before this then their antivirus is out-of-date. (The first observation in the test data is September 27, so you use this difference when encoding the test data.)

driveA is the ratio of hard drive partition used for the operating system with the total hard drive. Savy users install multiple operating systems and have a lower ratio. Savy users have reduced HasDetections.

driveB is the difference between hard drive partition used for the operating system and total hard drive. Responsible users' manager their hard drives well. Responsible users have reduced HasDetections.

7.2.2 Feature Encoding

In many practical data science activities, the data set will contain categorical variables. Since machine learning is based on mathematical equations, it would cause a problem when trying to keep categorical variables as is.

The following types of frequency encoding has been performed on the categorical variables made available to us by Microsoft:

- Frequency encoding:

It is a way to utilize the frequency of the categories as labels. In the cases where the frequency is related somewhat with the target variable, it helps the model to understand and assign the weight in direct and inverse proportion, depending on the nature of the data. There

are a lot of time dependent variables that exist like (EngineVersion, AvSigVersion, AppVersion, Census_OSVersion, Census_OSBuildRevision) .

And the test set would have variables from versions from beyond the time, the train and test variables are frequency encoded separately.

- One Hot Encoding:

In this method, mapping each category to a vector that contains 1 and 0 denoting the presence of the feature or not is done. The number of vectors depends on the categories which is wanted to keep. Among all our category variables, there are a combined 211,562 values. It is tedious to one-hot-encode all.

Then for each value, testing of the following hypotheses is done

$$H_0: \text{Prob (HasDetections=1 given value is present)} = 0.5 \quad \dots(7.1)$$

$$H_A: \text{Prob (HasDetections=1 given value is present)} \neq 0.5 \quad \dots(7.2)$$

The test statistic z-value equals \hat{p} , the observed HasDetections rate given value is present, minus 0.5 divided by the standard deviation of \hat{p} . The Central Limit Theorem tells us

$$Z\text{-value} = \hat{p} - 0.5 / \text{SD}(\hat{p}) \quad \dots(7.3)$$

If the absolute value of z is greater than 2.0, 95% confidence that Prob (HasDetections=1 given value is present) is not equal to 0.5 and the system will include a Boolean for this value in our model. Actually, we'll use a threshold of 5.0 and require $10^{-7}n > 0.005$. This adds lesser new Boolean variables.

Note that these feature encoding methods have only been used in LightGBM and recurrent neural network models. XDeepFM has its own factorization module.

7.3 Light Gradient Boosting Method

Boosting is a technique of making weak learners perform better by making modifications.

LGBM grows tree leaf-wise while other algorithm grows level-wise. Leaf-wise algorithm can reduce more loss than a level-wise algorithm. It can handle the large size of data and takes lower memory to run. It focuses on accuracy of results and supports GPU learning. It is sensitive to overfitting.

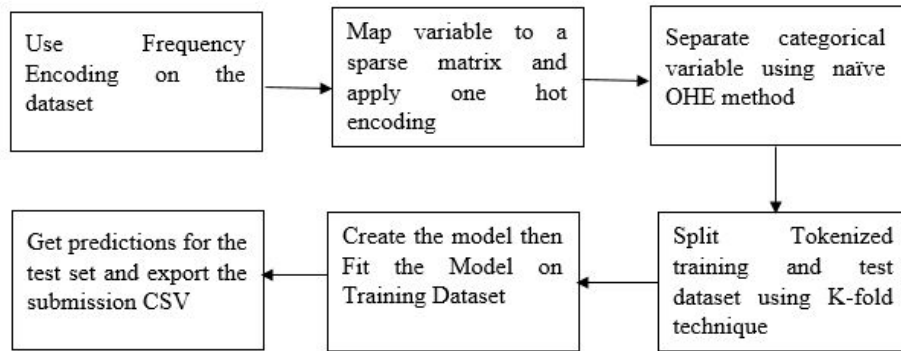


Figure 7.1: LGBM Process Flow illustrating the flow of data in the model

Figure 7.1 shows us the process flow of LGBM model. Clearly indicated in figure, the data must undergo preprocessing before it can be trained by the model. The training dataset is then converted into sparse matrix because of the huge number of null variables present in our dataset and it performs better in training. After this K fold cross validation is performed where the dataset is divided into k number of sets and then randomly its trained and tested. For our dataset $k = 10$. Now for LGBM firstly the parameters are decided by hyperparameter tuning.

7.3.1 Algorithm

```
Input: Labeled dataset (consisting of OS environment variables)
Output: Probability of that particular system OS build being affected by
malware
Frequency_encoding:
Count the frequency of all the variables separately and store the values
Code <- Divide the count by the maximum count to generalise.
Map the variable to the code now
Return code
One_Hot_Encoding:
category<-Choose category values that comprise more than 50% of and has
significance greater than "Z-value"
For i in category:
Return Separate the categorical variable 'i' as new columns according to the
same naive OHE method
Preprocessing_Dataset:
For i in Time_Series_Attributes
Frequency_encoding(i)
For i in every_other_data:
One_Hot_Encoding()
K <- 5
Training_Dataset, Test_Dataset <- Split Tokenized training and test dataset
using K-fold technique
File.h5 <- Store Training_Dataset, Test_Dataset as an h5 file to be used by
all models
Create the model with max_depth=1, n_estimators=30000, learning_rate=0.05,
Fit the Model on Training_Dataset
Training_history <- Save the training history as csv file
Get predictions for the test set and prepare a submission CSV
```

Figure 7.2: Pseudocode of the lightGBM model

The above figure 7.2 shows the complete procedure followed by the lightGBM model for our training set. The preprocessing of the training set, encoding phase and the final training scenario has been mentioned in the pseudocode. Note that the trained model is saved in an h5 format so that it can be reused during prediction and the model does not have to be trained

every time before prediction is made for a new system. The hyperparameters chosen for the particular model is mentioned as follows; `max_depth=1`, `n_estimators=30000`, `learning_rate=0.05` are the values used for lightGBM.

7.3.2 Console Output

```
-----
Transform Data to Sparse Matrix.
Sparse Matrix can be used to fit a lot of models, eg. XGBoost, LightGBM, Random Forest, K-Means and etc.
To concatenate Sparse Matrices by column use hstack()
Read more about Sparse Matrix https://docs.scipy.org/doc/scipy/reference/sparse.html
Good Luck!
-----

LightGBM

Fold 1

Training until validation scores don't improve for 100 rounds.
[100] valid_0's binary_logloss: 0.692329    valid_0's auc: 0.515657
Early stopping, best iteration is:
[1]   valid_0's binary_logloss: 0.693036    valid_0's auc: 0.515657
Fold 2

Training until validation scores don't improve for 100 rounds.
[100] valid_0's binary_logloss: 0.693624    valid_0's auc: 0.50221
Early stopping, best iteration is:
[1]   valid_0's binary_logloss: 0.693088    valid_0's auc: 0.50221
Fold 3

Training until validation scores don't improve for 100 rounds.
[100] valid_0's binary_logloss: 0.692962    valid_0's auc: 0.508201
Early stopping, best iteration is:
[1]   valid_0's binary_logloss: 0.693061    valid_0's auc: 0.508201
Fold 4

Training until validation scores don't improve for 100 rounds.
[100] valid_0's binary_logloss: 0.692039    valid_0's auc: 0.520162
Early stopping, best iteration is:
[1]   valid_0's binary_logloss: 0.693026    valid_0's auc: 0.520162
Fold 5

Training until validation scores don't improve for 100 rounds.
[100] valid_0's binary_logloss: 0.692256    valid_0's auc: 0.516667
Early stopping, best iteration is:
[1]   valid_0's binary_logloss: 0.693034    valid_0's auc: 0.516667
```

Figure 7.3: Console Output of LGBM Model showing AUC scores for each fold

The console output shown in figure 7.3 clearly shows the Accuracy scores per fold. The figure also indicates that the scores don't improve for 100 rounds. The accuracy is the highest at the 4th fold. It has a value of 0.520162.

7.4 XDeepFM

Extreme Deep Factorization Model utilizes deep learning along with a factorization machine. The model provided the highest accuracy among all the implemented models. It combines feature interactions, both implicit and explicit.

The model was primarily used for Recommender Systems. In recommender systems with a very large number of features to train, the results can get very skewed due to the huge number of influences the features have. In certain data sets, the dimension of the features is huge but the input features or actual usable features are very sparse. Other alternatives for classic recommender systems with sparse input features is logistic regression with “Follow the Regularized Leader”. XDeepFM provides a very high accuracy when it comes to cross features or multi-way features. These are features that combine categorical raw features to give very specific training scenarios. However, cross feature engineering has disadvantages. Getting significant information or trainable data from cross features is accompanied by a high cost. The computation is heavy and exploratory data analysis needs to find out specific patterns to choose the features to enable cross features. This is specific to each dataset and data scientists have their work cut out. In massive datasets with hundreds of features, it is not possible to extract all cross features manually to test for accuracy.

Finally, the patterns that data scientists pick out for cross featuring can miss out on many hidden or unseen features which may have been actually significant to improving the accuracy. Due to these problems, factorization machine is used. The machine embeds each feature in a latent vector to create pairwise interactions of every feature. The figure 7.4 shows how the embedding layer interacts with the input features. The hidden layers go through multiple processing functions after factorization.

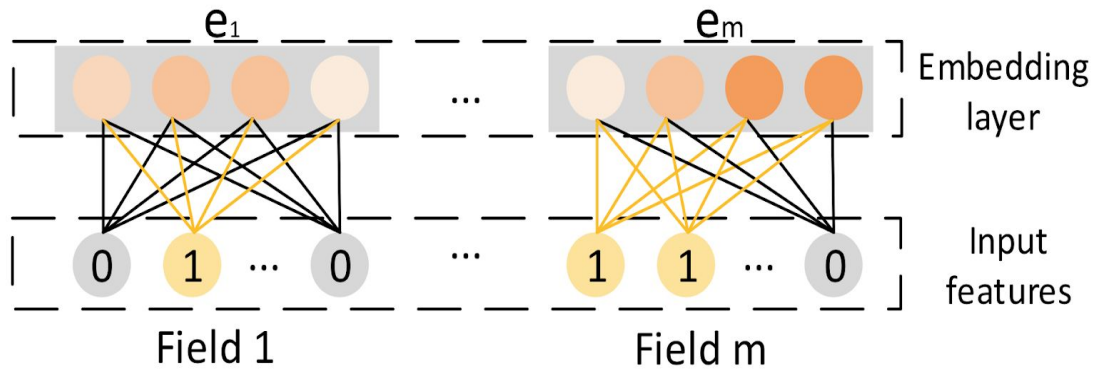


Figure 7.4: XDeepFM Model

7.4.1 Factorization Machine

The factorization machine is a vital part of XdeepFM. It helps us consolidate important features from the model rather than letting data scientists do EDA and hand pick the features. The machine compensates for scarcity of the input model and dataset.

The machine works by creating a matrix of n dimensions where n is the number of user inputs. However, majority of the matrix is zero as the input is scarce. If k is the number of features needed to be consolidated, cross multiplication of the n matrices to get the final matrix with embedded feature values is performed.

In our model, an embedding layer is applied on the raw input to compress the sparse matrix to a dense low dimensional matrix. This reduces the cost of training by a significant margin.

7.4.2 Compressed Interaction Network

While the factorization machine learns features implicitly, the Compressed Interaction Network learns other features explicitly and the extent of growth grows with the depth of the network. The interactions for the CIN will stay at a constant level of complexity and will not grow with interaction depth.

The Compressed Interaction Network takes features from both Convolutional Neural Networks and Recurrent Neural Networks.

Convolutional Neural Networks work based on moving a filter along the network layer. By taking the dot product, it consolidates the entire layer into a required dimension that can be controlled by the sliding window filter. When it is passed upon multiple layers, multiple

outputs that can be stacked on one another is the output. These outputs are called single depth slice. The CNN then has a very cost-effective method called pooling. Essentially, it takes a single depth layer and applies a logical relationship over an entire section of the depth. This ensures a lower dimension and easier calculation. It also reduces the number of parameters required.

In our model, the CIN does pooling at each logical layer to reduce the number of parameters. After the pooling is done, it passes the output as the input to the next layer which encompasses the basic structure of a Recurrent Neural Network.

With our dataset, the hidden layer size is a 128×128 matrix and the cross layer is a 3D matrix with the same dimensions. The learning rate is 0.001 so as to not overfit with just 1 epoch. This ensures efficiency of the model and at the same time, does not spend additional resources like LGBM on HasDetections.

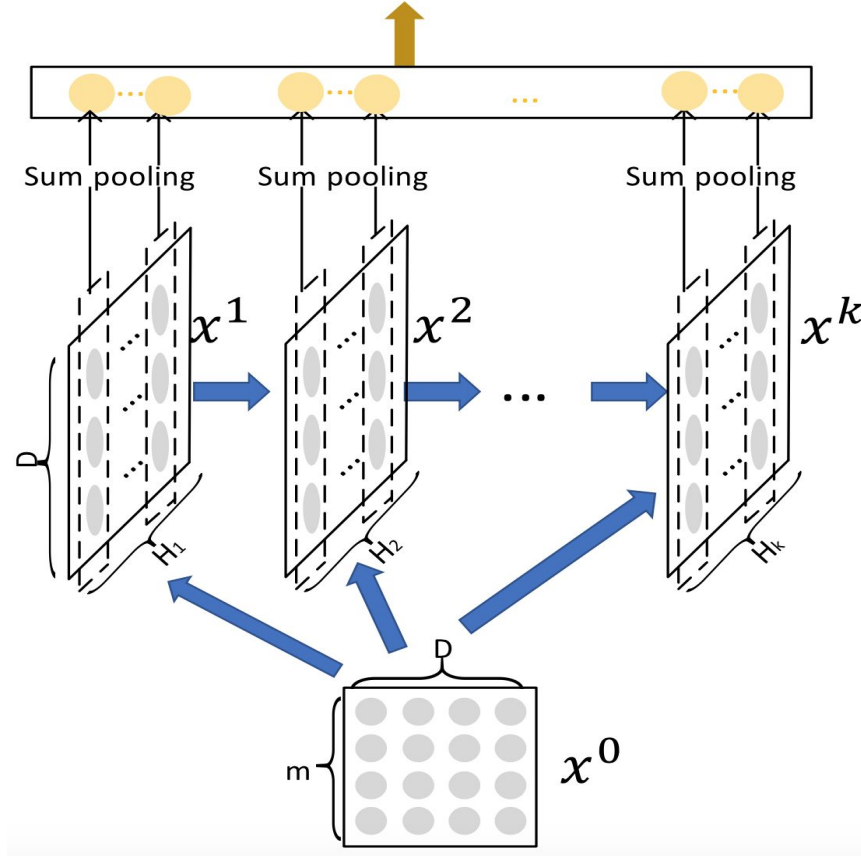


Figure 7.5: Compressed Interaction Network in XDeepFM

As seen in the figure 7.5, the compressed interaction network does sum pooling as a feature of convolutional neural network. The other component of the figure shows how it also takes the core of recurrent neural network in its core functioning.

7.4.3 Algorithm

```
Input: Labeled dataset (consisting of OS environment variables)
Output: Probability of that particular system OS build being affected by
malware
Frequency_encoding: Count the frequency of all the variables separately and
store the values
Code <- Divide the count by the maximum count to generalise Map the variable
to the code now
Return code
One_Hot_Encoding:
category<-Choose category values that comprise more than 50% of and has
significance greater than "Z-value"
For i in category:
Return Separate the categorical variable 'i' as new columns according to the
same naive OHE method
Preprocessing_Dataset:
For i in Time_Series_Attributes
Frequency_encoding(i)
For i in every_other_data:
One_Hot_Encoding()
K <- 5
Training_Dataset, Test_Dataset <- Split Tokenized training and test dataset
using K-fold technique
Factorization Machine - Multiplicative factorization of necessary features
with dimension n x k where n is features and k is required features
File.h5 <- Store Training_Dataset, Test_Dataset as an h5 file to be used by
all models
Create the model :
Model <- Add two layers of convolutional Neural Network to the model
Model <- Add necessary dropout values to ensure overfitting does not take
place
Model <- Add categorical_crossentropy as loss function and AUC scores as the
Accuracy measurement
Fit the Model on Training_Dataset
Training_history <- Save the training history as csv file
Get predictions for the test set and prepare a submission CSV
```

Figure 7.6: Pseudocode that illustrates the working of the xDeepFM model

The figure 7.6 shows the complete procedure followed by the XDeepFM model for the training set. The preprocessing of the training set, encoding phase and the final training scenario has been mentioned in the pseudocode. Note that the trained model is saved in a .h5 format so that it can be reused during prediction and the model does not have to be trained every time before prediction is made for a new system.

7.4.4 Console Output

```

Anaconda Prompt
# Epcho-time 82.03s Eval AUC 0.573770. Best AUC 0.573770.
# Epcho-time 83.48s Eval AUC 0.574214. Best AUC 0.574214.
'rm' is not recognized as an internal or external command,
operable program or batch file.
Training Done! Inference...
Fold 4
# Trainable variables
emb_v1:0, (200000, 1),
emb_v2:0, (200000, 8),
Variable:0, (648, 128),
norm_0/beta:0, (128,),
norm_0/gamma:0, (128,),
Variable_1:0, (128, 128),
norm_1/beta:0, (128,),
norm_1/gamma:0, (128,),
Variable_2:0, (128, 1),
exfm_part/f_0:0, (1, 6561, 128),
exfm_part/f_1:0, (1, 5184, 128),
exfm_part/f_2:0, (1, 5184, 128),
exfm_part/w_nn_output:0, (256, 1),
exfm_part/b_nn_output:0, (1,),
# Epcho-time 84.96s Eval AUC 0.589792. Best AUC 0.589792.
# Epcho-time 86.36s Eval AUC 0.591690. Best AUC 0.591690.
'rm' is not recognized as an internal or external command,
operable program or batch file.
Training Done! Inference...
0 0.488730
1 0.530526
2 0.500069
3 0.491964
4 0.520041
Name: HasDetections, dtype: float32
(base) C:\Users\Devika>

```

Figure 7.7: Console Output of XDeepFM showing Validation Accuracy.

Above Figure 7.7 shows that the best Accuracy Pre encoding is 0.574. It also shows the Epoch time taken for each fold. The time taken for the fold with best AUC score is 84.96 seconds.

The figure has one other important bit of information - all the trainable variables involved. Finally it also shows the vulnerability scores of the first 5 rows.

7.5 Recurrent Neural Networks

Recurrent Neural Networks or RNN are robust and powerful types of neural networks which belong to the most promising algorithms present in today's world because they are the only ones with an internal memory. Because of their internal memory, RNN's are able to remember important things about the input they received, which enables them to be very precise in predictions. Recurrent Neural Networks produce predictive results in sequential data that other algorithms cannot predict.

Unlike the Light Gradient Boosting framework model (LGBM), RNN is a strong learner. The RNN model used consists of a three-layer fully connected network with 100 neurons in each layer. The activation function used is ReLU. The Backpropagation dropout in the model used is forty percent.

Like the other two models, the pre-processing for the RNN model is done using One Hot Encoding and K-Fold Cross Validation. As per the accuracy scores between the three models, the RNN model was found to be the most accurate with a Training Accuracy of 0.68 (epoch 15) and a Validation Accuracy of 0.6 (Metric Used: ROC-AUC scores).

The disadvantage of this model is the high possibility of gradient vanishing or explosion when using the activation function.

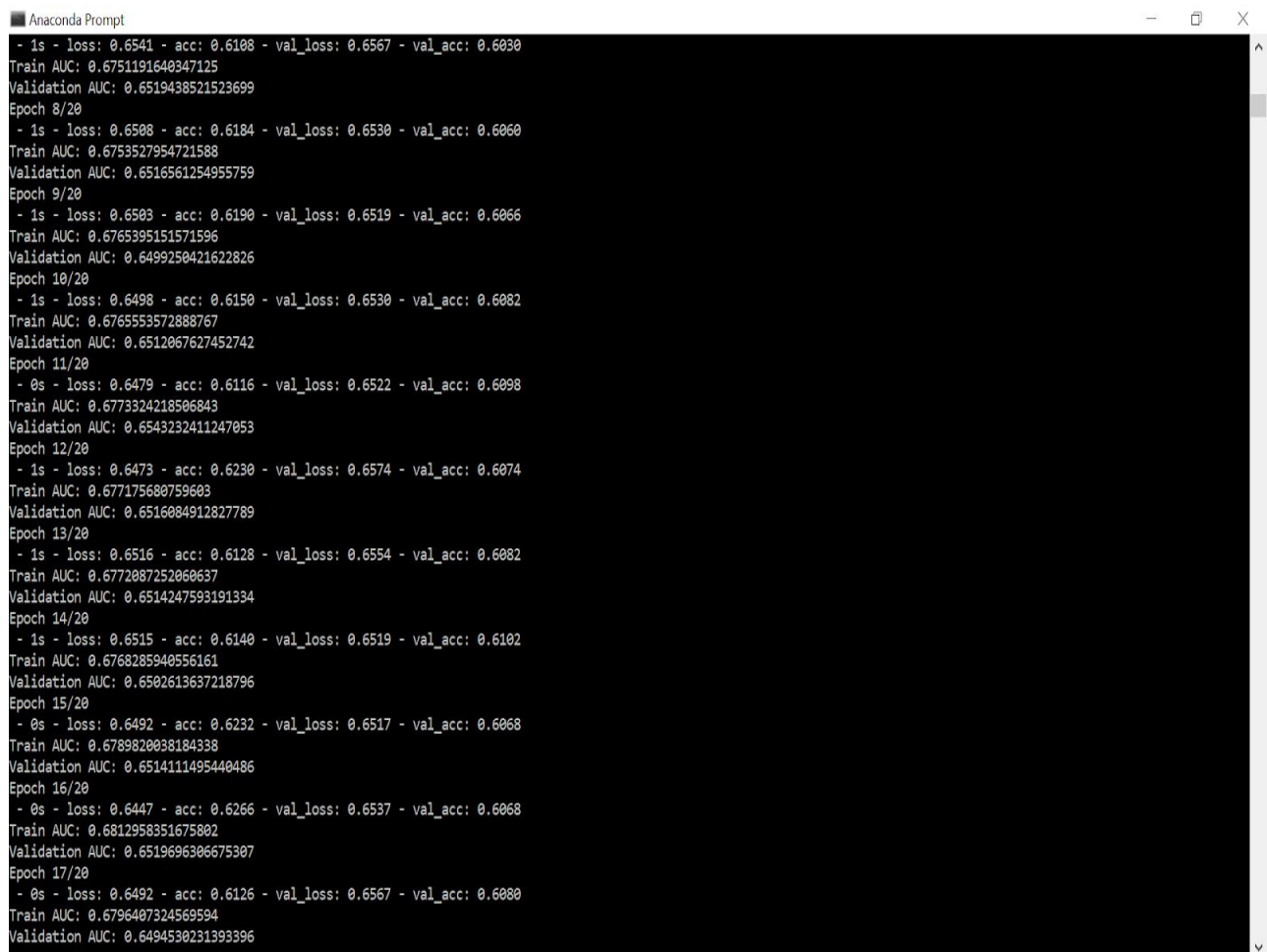
7.5.1 Algorithm

```
Input: Labeled dataset (consisting of OS environment variables)
Output: Probability of that particular system OS build being affected by
malware
Frequency_encoding:
Count the frequency of all the variables separately and store the values
Code <- Divide the count by the maximum count to generalize Map the variable
to the code now
Return code
One_Hot_Encoding:
category<-Choose category values that comprise more than 50% of and has
significance greater than "Z-value"
For i in category:
Return Separate the categorical variable 'i' as new columns according to the
same naive OHE method
Preprocessing_Dataset:
For i in Time_Series_Attributes
Frequency_encoding(i)
For i in every_other_data:
One_Hot_Encoding()
K <- 5
Training_Dataset, Test_Dataset <- Split Tokenized training and test dataset
using K-fold technique
File.h5 <- Store Training_Dataset, Test_Dataset as an h5 file to be used by
all models
Create the model :
Model <- Add two layers of Recurrent Neural Network to the model
Model <- Add necessary dropout values to ensure overfitting does not take
place
Model <- Add categorical_crossentropy as loss function and AUC scores as the
Accuracy measurement
Fit the Model on Training_Dataset
Training_history <- Save the training history as csv file
Get predictions for the test set and prepare a submission CSV
```

Figure 7.8: Pseudocode that illustrates the working of the recurrent model

The figure 7.8 shows the complete procedure followed by the recurrent neural network model for our training set. The preprocessing of the training set, encoding phase and the final training scenario has been mentioned in the pseudocode. Note that the trained model is saved in a .h5 format so that it can be reused during prediction and the model does not have to be trained every time before prediction is made for a new system. The hyperparameters chosen for the particular model is mentioned as follows; number of layers:2, dropout:0.02, k=10, learning_rate=0.05 are the values used for recurrent neural network.

7.5.2 Console Output



```
Anaconda Prompt
- 1s - loss: 0.6541 - acc: 0.6108 - val_loss: 0.6567 - val_acc: 0.6030
Train AUC: 0.6751191640347125
Validation AUC: 0.6519438521523699
Epoch 8/20
- 1s - loss: 0.6508 - acc: 0.6184 - val_loss: 0.6530 - val_acc: 0.6060
Train AUC: 0.6753527954721588
Validation AUC: 0.6516561254955759
Epoch 9/20
- 1s - loss: 0.6503 - acc: 0.6190 - val_loss: 0.6519 - val_acc: 0.6066
Train AUC: 0.6765395151571596
Validation AUC: 0.6499250421622826
Epoch 10/20
- 1s - loss: 0.6498 - acc: 0.6150 - val_loss: 0.6530 - val_acc: 0.6082
Train AUC: 0.6765553572888767
Validation AUC: 0.6512067627452742
Epoch 11/20
- 0s - loss: 0.6479 - acc: 0.6116 - val_loss: 0.6522 - val_acc: 0.6098
Train AUC: 0.6773324218506843
Validation AUC: 0.6543232411247053
Epoch 12/20
- 1s - loss: 0.6473 - acc: 0.6230 - val_loss: 0.6574 - val_acc: 0.6074
Train AUC: 0.677175680759603
Validation AUC: 0.6516084912827789
Epoch 13/20
- 1s - loss: 0.6516 - acc: 0.6128 - val_loss: 0.6554 - val_acc: 0.6082
Train AUC: 0.6772087252060637
Validation AUC: 0.6514247593191334
Epoch 14/20
- 1s - loss: 0.6515 - acc: 0.6140 - val_loss: 0.6519 - val_acc: 0.6102
Train AUC: 0.6768285940556161
Validation AUC: 0.6502613637218796
Epoch 15/20
- 0s - loss: 0.6492 - acc: 0.6232 - val_loss: 0.6517 - val_acc: 0.6068
Train AUC: 0.6789820038184338
Validation AUC: 0.6514111495440486
Epoch 16/20
- 0s - loss: 0.6447 - acc: 0.6266 - val_loss: 0.6537 - val_acc: 0.6068
Train AUC: 0.6812958351675802
Validation AUC: 0.6519696306675307
Epoch 17/20
- 0s - loss: 0.6492 - acc: 0.6126 - val_loss: 0.6567 - val_acc: 0.6080
Train AUC: 0.6796407324569594
Validation AUC: 0.6494530231393396
```

Figure 7.9: Console Output of Recurrent Neural Network and it's accuracy at each epoch.

In the figure 7.9, the console output of recurrent neural net training is shown. The first bit of information that can be drawn from the figure is that the model's accuracy scores from epoch 8 through 17 are highlighted. The accuracy is highest at epoch 12 where the validation accuracy score is 0.6074.

7.6 Pseudo Code (Main Module)

```
Input: OS build parameters obtained from the system from the landing screen
Output: the screen containing the percentage of vulnerability and the all
the properties displayed in report screen
Frequency_encoding:
Count the frequency of all the variables separately and store the values
Code <- Divide the count by the maximum count to generalize Map the
variable to the code now
Return code
One_Hot_Encoding:
category<-Choose category values that comprise more than 50% of and has
significance greater than "Z-value"
For i in category:
Return Separate the categorical variable 'i' as new columns according to
the same naive OHE method
Preprocessing_Dataset:
For i in Time_Series_Attributes
Frequency_encoding(i)
For i in every_other_data:
One_Hot_Encoding()
Render flask connection between the front end and back end
test<-Convert the OS build parameters into csv format
Encode the contents of the csv accordingly
model<-Load the model that was stored as .h5 file
Predict (model,test)
Add result to csv
Result<- read from csv and convert to dict
Render HTML CSS PAGE passing the Result
Display the screen
```

Figure 7.10: Pseudocode that illustrates the working of the final integrated project

The inference drawn from the figure 7.10 of the algorithm of the entire system is that the model is connected to the frontend via a server whose routes and API endpoints are created using a flask server. The server fetches the trained model details from an h5 file which stores

the model itself. The input is the OS build information obtained from the user's system. This is used to predict vulnerability using the h5 model. The predicted value is returned to the results html page.

7.7 Output

```
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
Using TensorFlow backend.
Loaded 10499 rows of TRAIN.CSV!
Only using 10499 rows to train and validate
FE encoded EngineVersion
FE encoded AppVersion
FE encoded AvSigVersion
FE encoded Census_OSVersion
OHE encoded RtpStateBitfield - Created 0 booleans
OHE encoded IsSxsPassiveMode - Created 0 booleans
OHE encoded DefaultBrowsersIdentifier - Created 0 booleans
OHE encoded AVProductStatesIdentifier - Created 2 booleans
OHE encoded AVProductsInstalled - Created 3 booleans
OHE encoded AVProductsEnabled - Created 0 booleans
OHE encoded CountryIdentifier - Created 0 booleans
OHE encoded CityIdentifier - Created 0 booleans
OHE encoded GeoNameIdentifier - Created 0 booleans
OHE encoded LocaleEnglishNameIdentifier - Created 0 booleans
OHE encoded Processor - Created 1 booleans
OHE encoded OsBuild - Created 0 booleans
OHE encoded OsSuite - Created 0 booleans
OHE encoded SmartScreen - Created 2 booleans
OHE encoded Census_MDC2FormFactor - Created 1 booleans
OHE encoded Census_OEMNameIdentifier - Created 0 booleans
OHE encoded Census_ProcessorCoreCount - Created 0 booleans
OHE encoded Census_ProcessorModelIdentifier - Created 0 booleans
OHE encoded Census_PrimaryDiskTotalCapacity - Created 0 booleans
OHE encoded Census_PrimaryDiskTypeName - Created 0 booleans
OHE encoded Census_HasOpticalDiskDrive - Created 0 booleans
OHE encoded Census_TotalPhysicalRAM - Created 2 booleans
OHE encoded Census_ChassisTypeName - Created 0 booleans
OHE encoded Census_InternalPrimaryDiagonalDisplaySizeInInches - Created 0 booleans
OHE encoded Census_InternalPrimaryDisplayResolutionHorizontal - Created 0 booleans
OHE encoded Census_InternalPrimaryDisplayResolutionVertical - Created 1 booleans
OHE encoded Census_PowerPlatformRoleName - Created 1 booleans
OHE encoded Census_InternalBatteryType - Created 0 booleans
OHE encoded Census_InternalBatteryNumberOfCharges - Created 0 booleans
OHE encoded Census_OSEdition - Created 0 booleans
OHE encoded Census_OSInstallLanguageIdentifier - Created 0 booleans
OHE encoded Census_GenuineStateName - Created 0 booleans
OHE encoded Census_ActivationChannel - Created 0 booleans
OHE encoded Census_FirmwareManufacturerIdentifier - Created 0 booleans
OHE encoded Census_IsTouchEnabled - Created 0 booleans
OHE encoded Census_IsPenCapable - Created 0 booleans
OHE encoded Census_IsAlwaysOnAlwaysConnectedCapable - Created 1 booleans
OHE encoded Wdft_IsGamer - Created 0 booleans
OHE encoded Wdft_RegionIdentifier - Created 0 booleans
Encoded 18 new variables
Removed original 43 variables
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Figure 7.11: Console Output of Main Module on Flask Server

The figure 7.11 shows console output when the flask server is run. It indicates that the system is using tensorflow for backend. Pre Processing is shown clearly where One Hot Encoding and its creation of 18 new variables are highlighted. The conversion from categorical to boolean for the variables using One Hot Encoding is a vital step. Finally, the figure says the flask server is running on 0.0.0.0:5000. 0.0.0.0 indicates the flask server is not running locally. Rather it is set to serve the application on the elastic IP address of the machine. 5000 indicates the port number of the application.

CHAPTER 8

TESTING

8.1 Introduction

In order for the web application to run seamlessly, certain test cases must be developed. These test cases will ensure that all the required conditions are met and prevents the application from throwing any errors or running infinite loops which would waste the user's time and resources.

8.2 Testing Tools and Environment

The test cases were written in Python 3.4 similar to the entire application. The test cases can be run on any environment compatible with Python and which also has Flask, nltk and all the other mentioned libraries installed and configured.

Two kinds of testing performed were unit testing and integration testing. Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing. While integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

8.3 Test cases

Table 8.1 shows us all unit testing for the modules conducted as part of the testing process. The table shows what conditions tests were run for. These tests are important to ensure a seamless user interface experience. The second column in the table shows the actions that the system will perform if there is an error when the condition is checked.

Table 8.1: Unit test cases for various modules of the project

Check Conditions	Action to be Performed
Check if OS Parameters are obtained before prediction	It should throw an appropriate error message showing that parameters have not been obtained successfully.
Check if one particular mandatory parameter is missing	Display message stating which parameter is missing.
Check for invalid parameters being sent	Display appropriate error message stating which parameter and the valid range of values.
Low internet connectivity	Displaying a message to inform the user of low internet connectivity
Check if accuracy values are correctly displayed	Verify the accuracy with the model used
Check that the values are randomised and taken every time	Verify by running the reading of values multiple times

Table 8.2 describes functional testing. These tests are performed to check inter module communication and functionality. The payload is what the system expects from the user as input. On receiving the payload, the system is expected to perform its operation and give resultant output. This output is shown in expected outcome.

Table 8.2: Functional/Integration testing

Details	Payload	Precondition	Step	Expected Outcome
To enter the URL for a first time user	URL = (string)	Good Internet Connectivity	1. Connect to the WIFI 2. Open a up-to-date browser 3. Fill in the URL in the search bar	The landing screen should appear with the visible scan button being pointed to.
To scan the system for the very first time	CTA button named "SCAN"	The browser should be on the landing screen by performing the previous step	1. Click on the scan button and wait to load	The scan in progress screen should appear.
To test the Scan-in-progress screen	Empty Payload	Previous step should be performed	Try to click or enter information or minimise during the scan	Should not get interrupted. Silent scan should take place in the backend. Make sure the model is being run in the

				backend.
To test the results screen	Empty payload	Previous step should be performed	Try to click or enter information or minimise during the scan	Should not change or vary the information displayed.
Reload the page	Reload CTA button	-	Click on the reload button	Should take it back to the landing screen
To scan the system for the second time	CTA button named "SCAN"	The browser should be on the landing screen by performing the previous step	1. Click on the scan button and wait to load	<p>The scan in progress screen should appear.</p> <p>Note that the results that appear should not vary more than +0.5/-0.5</p>

CHAPTER 9

RESULTS AND PERFORMANCE ANALYSIS

9.1 Result Snapshots

The following figure 9.1 is the landing screen of the final product achieved which is obtained by providing URL in the browser of any machine. The title and the logo clearly indicate the purpose of the product and the intended consumer branch. A simple easy to handle button is provided to begin the scanning of the operating system and to fetch the details required for the model to predict malware vulnerability of the system.

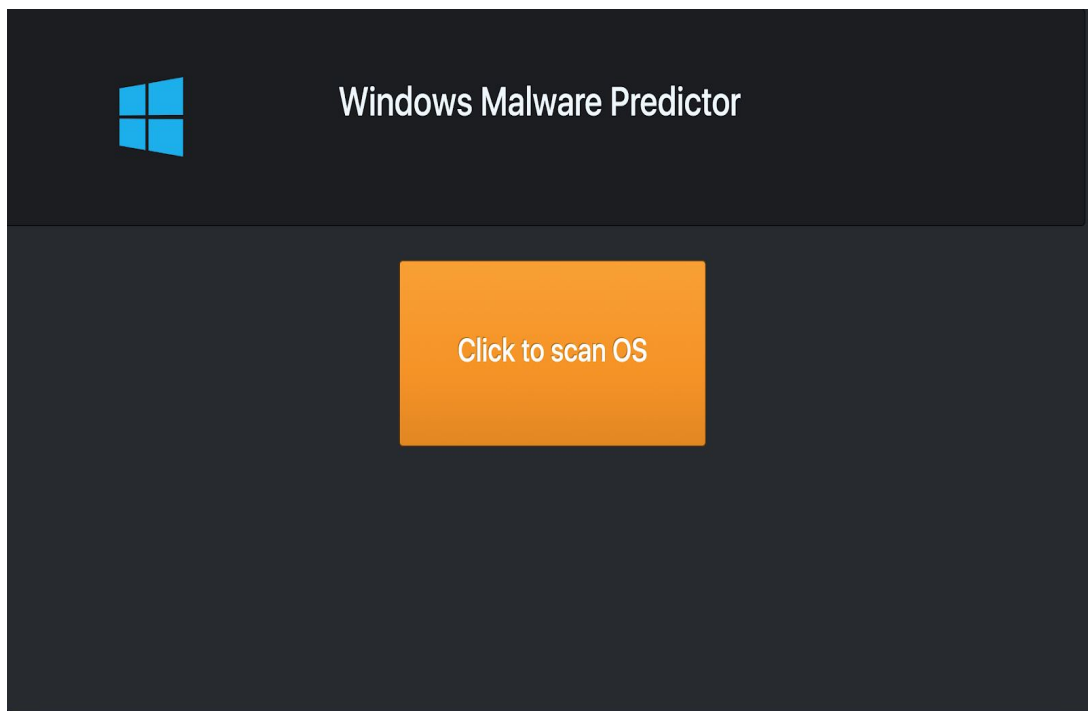


Figure 9.1: Landing Screen of the Malware Predictor

The CTA button leads to the following screen showing the fetching of data for the model. Figure 9.2 shows the scanning process on the GUI which is when the system fetches the data.



Figure 9.2: Scanning screen of the Malware Predictor

The browser then takes the user to the following page after the machine attributes were analysed by the machine learning model. This screen as shown in the above figure 9.3 contains the information that machine has been successfully scanned, the percentage of vulnerability: the intensity of the vulnerability is indicated by the colour of the text, and finally all the system properties that were provided as input to the machine learning model.

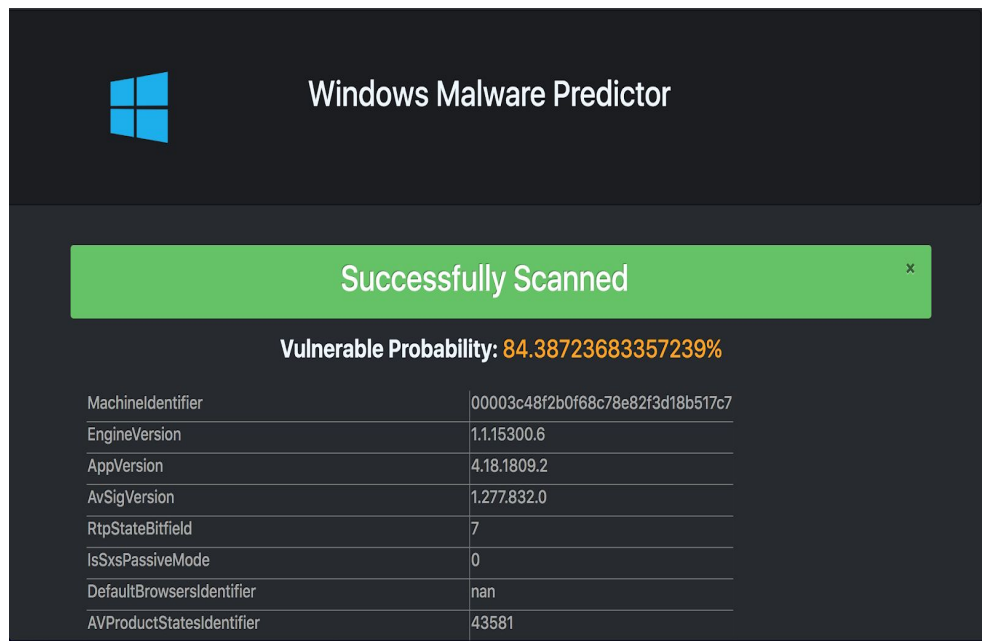


Figure 9.3: Result screen showing vulnerability of the Operating System

9.2 Comparison Tabular Results

The trained model is evaluated using three evaluation metrics - Validation Accuracy, Validation Loss, Training accuracy and Training Loss, calculated using ROC-AUC metric.

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN} \quad \dots(9.1)$$

False Positive Rate (FPR) is defined as follows:

$$FPR = \frac{FP}{FP + TN} \quad \dots(9.2)$$

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure 9.4 shows a typical ROC curve. As per the figure, the threshold for ROC curve is also illustrated at different points.

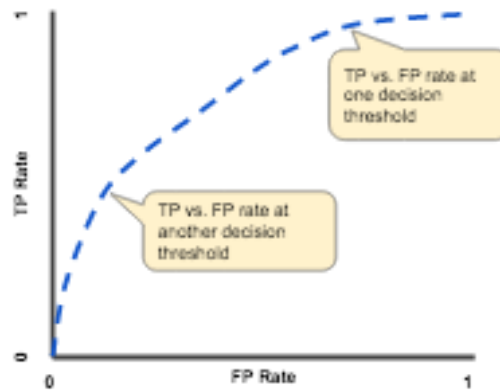


Figure 9.4: ROC-AUC curve

To compute the points in an ROC curve, evaluation could be done to evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is defined as the probability that the model ranks a random positive example more highly than a random negative example. AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

Table 9.1: LightGBM Results

Fold	Training Loss	Training AUC	Validation Loss	Validation AUC
1	0.692329	0.515657	0.693036	0.515532
2	0.693624	0.50221	0.693088	0.50213
3	0.692962	0.508201	0.693061	0.508194
4	0.692039	0.520162	0.693026	0.520123
5	0.693034	0.516667	0.693034	0.516452

Table 9.1 shows the results obtained by LightGBM on the dataset. The model was trained for 5 folds. The training accuracy is found to remain the same from the 1st fold till the 3rd fold. The maximum training and validation accuracy is obtained at the 4th fold. The training accuracy has a value of 0.520162 at the 4th fold. The least validation and training loss is also obtained at the 4th fold.

Table 9.2: XDeepFM Results

Fold	Validation Loss	Validation AUC	Training Loss	Training AUC
1	0.5981	0.7373	0.6225	0.718
2	0.6065	0.7276	0.5981	0.737
3	0.6213	0.7215	0.598	0.7383
4	0.6238	0.721	0.597	0.7389

Table 9.2 shows the results obtained by XDeepFM on the dataset. The XDeepFM was trained for 5 folds. The training accuracy is found to have a steady growth throughout. The maximum validation accuracy is obtained at the 4th fold at a value of 0.7389. The least training loss is obtained at the 4th fold too.

Table 9.3: Recurrent Neural Network results

Epoch	Loss	Accuracy	Validation Accuracy	Validation Loss	Training AUC	Validation AUC
1	0.6899	0.5736	0.6563	0.587	0.6654	0.6369
2	0.6596	0.598	0.6549	0.5976	0.667	0.6486
3	0.6608	0.5954	0.6545	0.6022	0.6697	0.6478
4	0.6557	0.6028	0.6517	0.602	0.6669	0.6444
5	0.652	0.6074	0.6518	0.6012	0.671	0.6509
6	0.6524	0.6086	0.6565	0.5902	0.6716	0.6494
7	0.6541	0.6108	0.6567	0.603	0.6751	0.6519
8	0.6508	0.6184	0.653	0.606	0.6753	0.6516
9	0.6503	0.619	0.6519	0.6066	0.6755	0.6499
10	0.6498	0.615	0.653	0.6082	0.6765	0.6512
11	0.6479	0.6116	0.6522	0.6098	0.6773	0.6543
12	0.6473	0.623	0.6574	0.6074	0.6771	0.6516
13	0.6516	0.6128	0.6554	0.6082	0.6772	0.6514
14	0.6515	0.614	0.6519	0.6102	0.6768	0.6502
15	0.6492	0.6232	0.6517	0.6068	0.6789	0.6514
16	0.6447	0.6266	0.6537	0.6068	0.6812	0.6519
17	0.6492	0.6126	0.6567	0.608	0.6796	0.6494
18	0.648	0.616	0.6519	0.6096	0.6787	0.6497
19	0.6441	0.6216	0.651	0.608	0.6835	0.6528
20	0.6472	0.62	0.6532	0.6028	0.6827	0.6505

Table 9.3 shows the results obtained by the recurrent neural network on the dataset. This model was trained for 20 epochs. The training accuracy is found to have a steady growth till the 13th epoch after which it drops sporadically.

9.3 Performance Analysis Graph

The below graph 9.5 depicts the behavior of the three different models of 1000 test data values. It is clearly observable from the figure that the LGBM model is incapable of differentiating between the vulnerable and least vulnerable machines and plays it safe by predicting all values around 50 percent. Neural network model is able to understand the extreme cases well but it uses a lot of resources and is time consuming. XDeepFM also

identifies a similar pattern but with much more efficiency as it requires lesser memory/CPU requirements and also takes the least time to train.

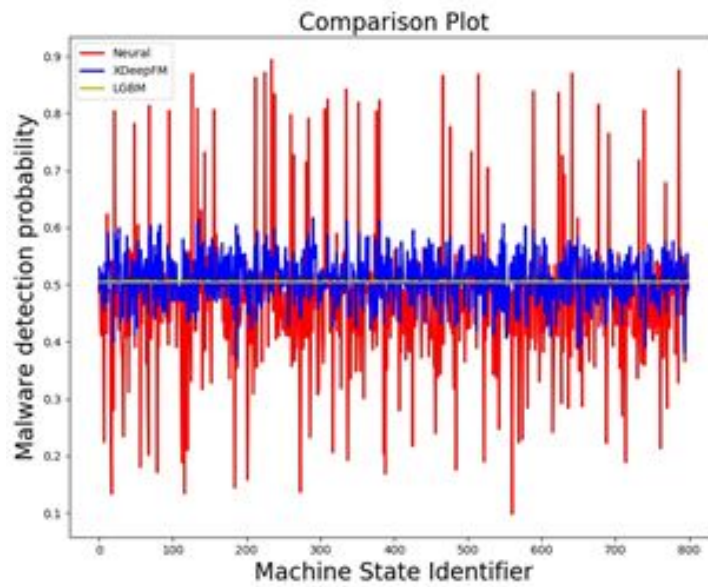


Figure 9.5 Model Behavior Comparison Plot

CONCLUSION AND FUTURE SCOPE

The project intends to build a model for predicting malware existence in operating systems with an impeccable accuracy. The outcome will also contain a hassle free user interface for the same as well. Future work could comprise of combining the works of detecting malware from the system codes written by understanding their semantics to help predict. Classification could also be done along with predictions. Malware can be very devastating depending on the target entity. Key-loggers can gain confidential information on entire organizations which can have monetary impact of billions of dollars. Viruses like Stuxnet can change the fate of an entire country by altering war. Catching a virus before it hits a system will improve efficiency and simplify code base of various anti-virus deployments.

Future work could comprise of combining the works of detecting malware from the system codes written by understanding their semantics to help predict.

REFERENCES

- [1] Baezner, Marie, and Patrice Robin. Stuxnet. No. 4. ETH Zurich, 2017.
- [2] You, Ilsun, and Kangbin Yim. "Malware obfuscation techniques: A brief survey." In 2010 International conference on broadband, wireless computing, communication and applications, pp. 297-300. IEEE, 2010.
- [3] Bethencourt, John, Dawn Xiaodong Song and Brent Waters. "Analysis-Resistant Malware". In: NDSS (2008)
- [4] Ekta Gandotra, Divya Bansal, Sanjeev Sofat (2016) "Zero-Day Malware Detection". In: 2016 Sixth International Symposium on Embedded Computing and System Design (ISED).
- [5] Miramirkhani, Najmeh, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. "Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts." In 2017 IEEE Symposium on Security and Privacy (SP), pp. 1009-1024. IEEE, 2017.
- [6] Rastogi, Vaibhav, Yan Chen, and Xuxian Jiang. "Catch me if you can: Evaluating android anti-malware against transformation attacks." IEEE Transactions on Information Forensics and Security 9, no. 1 (2014): 99-108.
- [7] Christodorescu, Mihai, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. "Semantics-aware malware detection." In 2005 IEEE Symposium on Security and Privacy (S&P'05), pp. 32-46. IEEE, 2005.
- [8] Burguera, Iker, Urko Zurutuza, and Simin Nadjm-Tehrani. "Crowdroid: behavior-based malware detection system for android." In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15-26. ACM, 2011.
- [9] Borbely, Rebecca Schuller. "On normalized compression distance and large malware." Journal of Computer Virology and Hacking Techniques 12, no. 4 (2016): 235-242.
- [10] Burnaev, Evgeny, and Dmitry Smolyakov. "One-class SVM with privileged information and its application to malware detection." In 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), pp. 273-280. IEEE, 2016.
- [11] Bhattacharya, Sukriti, Héctor D. Menéndez, Earl Barr, and David Clark. "Itect: Scalable information theoretic similarity for malware detection." arXiv preprint arXiv:1609.02404 (2016).

- [12] Jhonattan J Barriga, Sang Guun Yoo. "Malware Detection and Evasion with Machine Learning Techniques: A Survey" International Journal of Applied Research 2017
- [13] Hardy, William, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. "DL4MD: A deep learning framework for intelligent malware detection." In Proceedings of the International Conference on Data Mining (DMIN), p. 61. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.
- [14] Kolosnjaji, Bojan, Apostolis Zarras, George Webster, and Claudia Eckert. "Deep learning for classification of malware system call sequences." In Australasian Joint Conference on Artificial Intelligence, pp. 137-149. Springer, Cham, 2016.
- [15] Bagga, Naman. "Measuring the Effectiveness of Generic Malware Models." (2017).
- [16] Danny Hendler, Shay Kels and Amir Rubin (2018) "Detecting Malicious PowerShell Commands using Deep Neural Networks". In: arXiv:1804.04177, Cornell University, 2018.
- [17] Zhou H. (2019) Malware Detection with Neural Network Using Combined Features. In: Yun X. et al. (eds) Cyber Security. CNCERT 2018. Communications in Computer and Information Science, vol 970. Springer, Singapore.
- [18] Kris Kendall "Practical Malware Analysis" in BlackHat (2015)
- [19] Wagner, Markus, Fabian Fischer, Robert Luh, Andrea Haberson, Alexander Rind, Daniel A. Keim and Wolfgang Aigner. "A Survey of Visualization Systems for Malware Analysis." *EuroVis* (2015)
- [20] Bayer, Ulrich, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Krügel and Engin Kirda. "Scalable, Behavior-Based Malware Clustering." *NDSS* (2009).
- [21] Ye, Yanfang, Dingding Wang, Tao Li and Dongyi Ye. "IMDS: intelligent malware detection system." *KDD* (2007).
- [22] Rieck, Konrad, Thorsten Holz, Carsten Willems, Patrick Düssel and Pavel Laskov. "Learning and Classification of Malware Behavior." *DIMVA* (2008).
- [23] Griffin, Kent, Scott Schneider, Xin Hu and Tzi-cker Chiueh. "Automatic Generation of String Signatures for Malware Detection." *RAID* (2009).