

2021

A defensive strategy for detecting targeted adversarial poisoning attacks in machine learning trained malware detection models

Adrian Michael Wood
Edith Cowan University

Follow this and additional works at: <https://ro.ecu.edu.au/theses>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Wood, A. M. (2021). *A defensive strategy for detecting targeted adversarial poisoning attacks in machine learning trained malware detection models*. Edith Cowan University. Retrieved from <https://ro.ecu.edu.au/theses/2483>

This Thesis is posted at Research Online.
<https://ro.ecu.edu.au/theses/2483>

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

A Defensive Strategy for Detecting Targeted Adversarial Poisoning Attacks in Machine Learning Trained Malware Detection Models

A thesis presented for the degree of

Doctor of Philosophy

Adrian Michael Wood

School of Science

Edith Cowan University

Perth, Western Australia

2021

Abstract

Machine learning is a subset of Artificial Intelligence which is utilised in a variety of different fields to increase productivity, reduce overheads, and simplify the work process through training machines to automatically perform a task. Machine learning has been implemented in many different fields such as medical science, information technology, finance, and cyber security. Machine learning algorithms build models which identify patterns within data, which when applied to new data, can map the input to an output with a high degree of accuracy. To build the machine learning model, a dataset comprised of appropriate examples is divided into training and testing sets. The training set is used by the machine learning algorithm to identify patterns within the data, which are used to make predictions on new data. The test set is used to evaluate the performance of the machine learning model.

These models are popular because they significantly improve the performance of technology through automation of feature detection which previously required human input. However, machine learning algorithms are susceptible to a variety of adversarial attacks, which allow an attacker to manipulate the machine learning model into performing an unwanted action, such as misclassifying data into the attackers desired class, or reducing the overall efficacy of the ML model. One current research area is that of malware detection. Malware detection relies on machine learning to detect previously unknown malware variants, without the need to manually reverse-engineer every suspicious file. Detection of Zero-day malware plays an important role in protecting systems generally but is particularly important in systems which manage critical infrastructure, as such systems often cannot be shut down to apply patches and thus must rely on network defence. In this research, a targeted adversarial poisoning attack was developed to allow Zero-day malware files, which were originally classified as malicious, to bypass detection by being misclassified as benign files. An adversarial poisoning attack occurs when an attacker can inject specifically-crafted samples into the training dataset which alters the training process to the desired outcome of the attacker. The targeted adversarial poisoning attack was performed by taking a random selection of the Zero-day file's import functions and injecting them into the benign training dataset. The targeted adversarial poisoning attack succeeded for both Multi-Layer Perceptron (MLP) and Decision Tree models without reducing the overall efficacy of the target model. A defensive strategy was developed for the targeted adversarial poisoning attack for the MLP models by examining the activation weights of the penultimate layer at test time. If the activation weights were outside the norm for the target (benign) class, the file is quarantined for further examination. It was found to be possible to identify on average 80% of the target Zero-day files from the combined targeted poisoning attacks by examining the activation weights of the neurons from the penultimate layer.

Acknowledgments

I would like to thank my supervisors A/Prof Mike Johnstone and Dr. Peter Hannay for all of their support and guidance throughout my PhD and also my Honours degree. I can say with certainty that without their help it would not have been possible for me to complete this thesis, and for that I am truly grateful.

Contents

Abstract.....	i
Acknowledgments.....	i
1 Introduction.....	2
1.1 Background.....	4
1.2 Significance.....	6
1.3 Purpose.....	7
1.4 Research Questions.....	8
1.5 Outline of Thesis.....	8
2 Literature Review.....	11
2.1 Malware	11
2.1.1 Malware Types, Platforms and Families.....	11
2.1.2 Malware Detection.....	14
2.1.3 Malware Obfuscation.....	16
2.2 Machine Learning Algorithms	17
2.2.1 Learning Approaches	18
2.2.2 Training Approaches.....	19
2.2.3 Data Curation	20
2.3 Data Labelling.....	21
2.3.1 Feature Engineering	21
2.4 Model Accuracy (Performance Measures).....	22
2.4.1 Confusion matrix.....	23
2.4.2 Accuracy, Specificity, Precision and Recall	23
2.4.3 Precision-Recall or PR Curve	24
2.4.4 ROC (Receiver Operating Characteristics) Curve	26
2.4.5 F1 score.....	27
2.4.6 Matthews Correlation Coefficient.....	28
2.5 Algorithm Overview	28

2.5.1	Artificial Neural Networks.....	28
2.5.2	Support Vector Machines.....	29
2.5.3	Decision Trees.....	30
2.5.4	Regression Models.....	31
2.5.5	Naïve Bayes	31
2.5.6	K-means Clustering.....	32
2.6	Machine Learning Applications.....	32
2.6.1	Critical Infrastructure	32
2.6.2	Computer Vision	33
2.6.3	Natural Language Processing.....	33
2.6.4	Cyber Security.....	33
2.6.5	Finance	34
2.6.6	Marketing	34
2.6.7	Healthcare	34
2.6.8	Personal Assistants.....	35
2.7	Adversarial Machine Learning.....	35
2.7.1	Adversarial Attacks.....	38
2.7.2	Adversarial Defence Strategies	47
2.8	EMBER Dataset.....	53
2.9	Summary	56
3	Research Methods and Design.....	59
3.1	Philosophical Systems	59
3.1.1	Scientific Research Paradigms	60
3.2	Research Design.....	66
3.2.1	Hypotheses	66
3.2.2	Research Phases	67
3.2.3	Materials	70
3.3	Risks and Limitations	70
3.4	Ethics.....	71

3.5	Summary	72
4	Experimentation – Adversarial Poisoning Attacks and Defences.....	73
4.1	Environment.....	73
4.2	Datasets.....	73
4.2.1	EMBER.....	74
4.2.2	VirusShare.....	75
4.3	Feature Analysis.....	75
4.4	Preliminary Experiments.....	78
4.4.1	Preliminary Experiments Stage One	80
4.4.2	Preliminary Experiments Stage Two	87
4.4.3	Preliminary Experiments Discussion	93
4.5	Main Experiments	97
4.5.1	Individual and Combined Attack Tests.....	98
4.6	Defensive Strategies.....	114
4.6.1	Heatmaps.....	115
4.6.2	Feature Importance	116
4.6.3	Activation Clustering	117
4.7	Summary	127
5	Results.....	128
5.1	Preliminary Results	128
5.2	Targeted Attack Results (GBDT)	130
5.3	Targeted Attack Results (MLP)	132
5.4	Defence Results	135
5.5	Defence Contribution.....	140
5.6	Summary	140
6	Discussion.....	141
6.1	Adversarial Attack Discussion.....	141
6.2	Adversarial Defence Discussion	143
6.2.1	Neuron Pruning.....	143

6.2.2	Activation Clustering	144
6.3	Adversarial Attack and Defence Comparison.....	145
6.4	Research Question Discussion	146
6.5	Summary	148
7	Conclusion	149
7.1	Research Overview	149
7.2	Summary of Contributions.....	151
7.3	Future Work	153
8	References.....	154
9	Appendix A – Machine Learning Model Code.....	162
10	Appendix B – Experiment Result Tables.....	163

List of Figures

FIGURE 1.1. - TAY TWEET AFTER BEING FED AN ASSORTMENT OF DATA. ADAPTED FROM (NASH, 2016).....	5
FIGURE 2.1 - ML PROCESS ADAPTED FROM GÉRON (2019)	17
FIGURE 2.2 - MODEL FITTING EXAMPLES ADAPTED FROM - SCIKIT (2014)	21
FIGURE 2.3 - CONFUSION MATRIX WITH ACCURACY	23
FIGURE 2.4 - PR CURVE EXAMPLE	25
FIGURE 2.5 - MULTI-CLASS PRECISION-RECALL CURVE EXAMPLE ADAPTED FROM SCIKIT	26
FIGURE 2.6 - ROC CURVE EXAMPLE	27
FIGURE 2.7 - DIAGRAM OF A FEEDFORWARD ANN ADAPTED FROM ("ARTIFICIAL NEURAL NETWORK WITH LAYER COLORING," 2013)	29
FIGURE 2.8 - OPTIMAL HYPERPLANE OF A LINEAR SVM. SUPPORT VECTORS ARE COLOURED BLUE. ADAPTED FROM HAYKIN (2008).....	30
FIGURE 2.9 - THE ARCHITECTURE OF MALGAN ADAPTED FROM HU AND TAN (2017)	41
FIGURE 2.10 - LABEL SANITISATION DEFENCE RESULTS ADAPTED FROM (PAUDICE, MUÑOZ-GONZÁLEZ, & LUPU, 2018)	53
FIGURE 3.1 - POSITIVIST RESEARCH PROCESS ADAPTED FROM (WILLIAMSON & JOHANSON, 2017).....	61
FIGURE 3.3 - EDGAR AND MANZ RESEARCH DECISION TREE PT.1 ADAPTED FROM (EDGAR & MANZ, 2017)	64
FIGURE 3.4 - RESEARCH DESIGN	69
FIGURE 4.1 - EXAMPLE OF A BADNET TRIGGER IMAGE ADAPTED FROM (GU ET AL., 2019)	98
FIGURE 4.2 – SINGLE DIMENSION HEATMAP REPRESENTATION OF ACTIVATION WEIGHTS	116
FIGURE 4.3 - ZERO-DAY ACTIVATION WEIGHTS.....	118
FIGURE 4.4 - ACTIVATION WEIGHTS OF ZERO-DAY CORRECT CLASSIFICATION - EXAMPLE 1	119
FIGURE 4.5 - ACTIVATION WEIGHTS OF ZERO-DAY CORRECT CLASSIFICATION - EXAMPLE 2	119
FIGURE 4.6 - ACTIVATION WEIGHTS OF ZERO-DAY SUCCESSFUL MIS-CLASSIFICATION - EXAMPLE 1.....	120
FIGURE 4.7 - ACTIVATION WEIGHTS OF ZERO-DAY SUCCESSFUL MIS-CLASSIFICATION - EXAMPLE 2.....	120
FIGURE 5.1 - ZERODAY FALSE NEGATIVE EXAMPLE 1	135

List of Tables

TABLE 1 - CONFUSION MATRIX DESCRIPTION ADAPTED FROM (JOHNSTONE & PEACOCK, 2020)	22
TABLE 2 - ADVERSARIAL ATTACK SPECTRUM. ADAPTED FROM HUANG ET AL. (2011)	36
TABLE 3 - STINGRAY ATTACK RESULTS (AVERAGE INSTANCES, SUCCESS RATE, AND PERFORMANCE DROP RATE) ADAPTED FROM (SUCIU ET AL., 2018)	40

TABLE 4 - MALGAN ADVERSARIAL ATTACK RESULTS (DIFFERENT TRAINING SET) ADAPTED FROM -HU AND TAN (2017).....	42
TABLE 5 - MALGAN ADVERSARIAL ATTACK RESULTS 9SAME TRAINING SET) ADAPTED FROM -HU AND TAN (2017)	42
TABLE 6 - WATERMARK ATTACK AND DEFENCE RESULTS ADAPTED FROM (SEVERI ET AL., 2020).....	46
TABLE 7 - ACTIVATION AND RAW IMAGE DATA CLUSTERING RESULTS ADAPTED FROM (B. CHEN ET AL., 2018)	50
TABLE 8 - MODEL PATCHING PERFORMANCE COMPARISON ADAPTED FROM (WANG ET AL., 2019).....	51
TABLE 9 - EMBER FEATURE ENGINEERING	54
TABLE 10 - REALTIME PE-MINER EXTRACTED PE FEATURES ADAPTED FROM (SHAFIQ ET AL., 2009).....	56
TABLE 11 - SUMMARY OF IDENTIFIED PROBLEMS	58
TABLE 12 - RESEARCH METHODS ADAPTED FROM WILLIAMSON AND JOHANSON (2017).....	60
TABLE 13 - INFORMATION SYSTEMS RESEARCH APPROACHES. ADAPTED FROM (GALLIERS, 1990).....	62
TABLE 14 - RESEARCH QUESTION RELATIONSHIPS	67
TABLE 15 - RISKS AND MITIGATION STRATEGIES	71
TABLE 16 - EMBER PERFORMANCE 1% FPR.....	74
TABLE 17 - EMBER PERFORMANCE 0.1% FPR.....	75
TABLE 18 - IMPORT LIBRARY AND FUNCTION AMOUNT	77
TABLE 19 - VIRUSSHARE_00352 IMPORT LIBRARY AND FUNCTION COUNT	79
TABLE 20 - TOP 20 IMPORT LIBRARIES	79
TABLE 21 - KERNEL32.DLL FIVE FUNCTIONS PRELIMINARY POISONING ATTACK (MANUAL SELECTION)	81
TABLE 22 - EXPERIMENT 1.1B INJECTION PERCENTAGES	82
TABLE 23 - MANUAL IMPORT SELECTION - 5% TOTAL BENIGN FEATURE SPACE	84
TABLE 24 - EXPERIMENT 2.1A INJECTION PERCENTAGES.....	85
TABLE 25 - EXPERIMENT 2.1B INJECTION PERCENTAGES.....	86
TABLE 26 - RANDOM IMPORT SELECTION - 5% TOTAL BENIGN FEATURE SPACE.....	87
TABLE 27 - EXPERIMENT 2.1A RESULTS	88
TABLE 28 - EXPERIMENT 2.1B RESULTS	89
TABLE 29 - BENIGN TRAINING FILES INJECTION PERCENTAGE - MANUAL SELECTION ATTACK	90
TABLE 30 - EXPERIMENT 2.1A RESULTS.....	91
TABLE 31 - EXPERIMENT 2.1B RESULTS	92
TABLE 32 - BENIGN TRAINING FILES INJECTION PERCENTAGE - RANDOM SELECTION ATTACK	93
TABLE 33 - GENERAL EFFICACY ATTACK - FEATURE SPACE MANUAL	94

TABLE 34 - H1, H3, AND H4 RESULTS.....	95
TABLE 35 - GENERAL EFFICACY ATTACK - FEATURE SPACE RANDOM	96
TABLE 36 - H ₂ , H ₃ AND, H ₄ EXPERIMENT RELATIONSHIP AND RESULTS.....	97
TABLE 37 - IMPORT FUNCTIONS - TARGETED ATTACK	100
TABLE 38 - 5% POISON ATTACK - FIRST TEN RESULTS	102
TABLE 39 - 25% POISON ATTACK - FIRST TEN RESULTS	103
TABLE 40 - INDIVIDUAL LIBRARY POISON ATTACK RESULTS	104
TABLE 41 - BENIGN TRAIN DATA CONTAINING ALL TARGET IMPORT LIBRARIES	105
TABLE 42 - COMBINED ATTACK RESULTS GBDT.....	105
TABLE 43 - COMBINED ATTACK GBDT - FIRST TEN RESULTS - SAME SEED.....	106
TABLE 44 - COMBINED ATTACK GBDT - FIRST TEN RESULTS - RANDOM SEED	107
TABLE 45 - MLP COMBINED ATTACK APPROACH - FIRST TEN RESULTS - SAME SEED	111
TABLE 46 - TARGETED ADVERSARIAL POISONING ATTACK RESULTS - MLP.....	113
TABLE 47 - RANDOMISED HYBRID AND COMBINED RESULTS COMPARISON	114
TABLE 48 - MAD NEURON DEFENCE EXAMPLES	122
TABLE 49 - INDIVIDUAL ATTACK (25%) DEFENCE RESULTS	123
TABLE 50 – HYBRID ATTACK (8.6%) DEFENCE RESULTS	123
TABLE 51 - HYBRID ATTACK RANDOMISED (8.6%) DEFENCE RESULTS	124
TABLE 52 - HYBRID ATTACK RANDOMISED (15%) DEFENCE RESULTS	124
TABLE 53 - COMBINED ATTACK RANDOMISED (10%) DEFENCE RESULTS	125
TABLE 54 - MAD DEFENCE EXPERIMENT - POISONED TEST DATA	125
TABLE 55 - KERNEL32.DLL - LAST THREE ATTACK RESULTS FROM EACH PRELIMINARY EXPERIMENT	129
TABLE 56 - H ₁ - H ₄ EXPERIMENT RELATIONSHIP AND RESULTS	129
TABLE 57 - TARGETED ADVERSARIAL POISONING ATTACK (INDIVIDUAL APPROACH) - GBDT RESULTS	130
TABLE 58 - TARGETED ADVERSARIAL POISONING ATTACK (COMBINED APPROACH) - GBDT RESULTS	131
TABLE 59 - MLP TARGETED ADVERSARIAL ATTACK RESULTS.....	134
TABLE 60 – MAD DEFENCE RESULTS SIX NEURONS	137
TABLE 61 - MAD DEFENCE RESULTS EIGHT NEURONS	138
TABLE 62 - MAD DEFENCE RESULTS TEN NEURONS	138
TABLE 63 - RESEARCH QUESTION RELATIONSHIPS.....	151

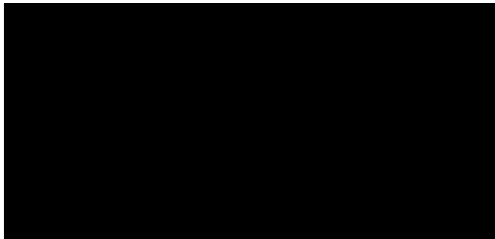
I certify that this thesis does not, to the best of my knowledge and belief:

i) incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;

ii) contain any material previously published or written by another person except where due reference is made in the text of this thesis;

iii) contain any defamatory material;

Signed:



Adrian Michael Wood

Dated: 11th June 2021

1 Introduction

In 2015 the global cost of cybercrime was US\$ 3 trillion which is estimated to grow to US\$ 6 trillion by 2021 (Cybersecurity Ventures & Herjavec Group, 2019). For large-sized Australian businesses, an average of US\$ 6.8 million was spent on cybercrime in 2018, up from US\$ 5.4 million in 2017 (Accenture, 2017). Malware was the most common form of attack reported, with the cost of malware attacks increasing by 11% between 2017-2018 (Accenture, 2019). Small to medium-sized enterprises (SMEs) are the most vulnerable group with respect to cyberattacks, as they lack the resources to put in place appropriate security measures needed for protection. SMEs worldwide spend on average less than US\$ 4,000 per year on cyber security protection (Juniper, 2017).

Out of the \$6.8 million dollars spent by larger Australian companies on mitigating cybercrime, discovery and containment contribute the majority (60%) of the cost, with discovery comprising 36% and containment 24% (Accenture, 2019). Mitigation comes at differing levels of cost and complexity, from security intelligence systems to automation and machine learning (ML). The most commonly used defensive systems are security intelligence systems (67%) and advanced identity and access governance (63%), while ML is the third least commonly deployed security technology (38%) but provides the second greatest cost savings of the aforementioned measures (Accenture, 2019).

The impact of cybercrime is not only focused on the financial aspects but also the operation of critical infrastructure, which nations rely upon to function. Critical infrastructure systems may rely upon more tailored and unique protection, due to the intricate nature of the systems. The complexity of the systems exacerbates the damage caused by an incident which increases the cost required for mitigation.

In cyber security, ML is seen as a promising tool with significant potential, but ML has some drawbacks which can negate its value. If an attacker has knowledge of the ML algorithm in use or can in some way gain access to the dataset being utilised to train the ML model, the attacker is then able to attack the ML system itself.

A benefit of ML in the application of cyber security is in the use of malware detection. Traditional malware detection applications rely upon signatures to identify if a software file is malicious. The signature is generated after the malicious properties of a software file have been identified, either by static or dynamic analysis and are then added to a repository of malicious signatures which anti-malware applications use to identify malware by scanning the file and comparing the signature. The drawback of the signature approach is that it can only identify malicious software which it already knows about (known-knowns), and it is unable to detect malicious software which has not already been identified (known-unknowns and unknown-unknowns). In the ML malware detection approach, the ML model has been trained to identify patterns within software files which distinguish them as

either malicious or benign. When the ML model is presented with a new software file, it can make a classification based upon the patterns within. It is important to note that ML models are not perfect, and false positives (a benign file being classified as malicious) and false negatives (a malicious file being classified as benign) will appear on occasion.

Malware detection is an area of cyber security which can reap the benefit of ML. The ML models generated for malware detection automate the process of determining if a file or network stream is of a malicious nature. For the training of ML-based malware detection applications, an open training model, where the datasets used are constantly being supplied with new files for further training, is essential to keep pace with the ever-increasing rate of malware creation. For Intrusion Detection Systems (IDSs), monitoring the flow of traffic and updating rule sets based on behaviour can help reduce false positives being reported as the machine learning model has learnt from the new network behaviour.

As malware detection relies on training with current malware data, a more complex problem occurs if an attacker is able to influence the training process by exploiting the open training model. It is possible that specifically-crafted adversarial malware could be created to infect and reduce the efficacy of ML-based malware detection applications which can lead to an increase of false negatives.

There is a need to research different ML algorithms in relation to how these algorithms can be utilised to detect Zero-day attacks and further, how adversarial attacks can reduce the efficacy of ML based malware/intrusion detection applications. For known-known malware files, detection is straightforward as the files have already been identified as malicious and a signature has been generated for future identification. Zero-day malware, which does not have a signature as the file has not yet been identified as malicious, requires other means of identification. One of the methods to identify Zero-day malware is through the use of ML trained malware detection models. The ML model identifies malware by recognising patterns within both malicious and benign software files, and classifies new files based upon the identified patterns in the new data.

When training ML models, the conventional approach is to assume that the data being used is trustworthy and accurately represents the area being investigated (Pitropakis, Panaousis, Giannetsos, Anastasiadis, & Loukas, 2019). In a malware detection ML application, one approach is to scan the training data using a multitude of anti-malware applications and label the data according to the majority of the results. In this approach, the researchers are trusting the output of the anti-malware applications to produce an accurate label for the input data. It is not usually considered that the files may have been modified by an attacker (without changing the functionality of the file) to influence the training of the ML model for some other outcome e.g., certain malicious files bypass detection while the overall classification rate is unchanged.

It is proposed in this thesis that evaluating various attacks against different ML algorithms would provide valuable insight in determining which algorithms are best suited for use in malware detection and what preventative measures may be implemented to harden the selected ML algorithms against adversarial attacks.

1.1 Background

Machine learning (ML) is a sub-field of Artificial Intelligence which has found application in many different domains. Machine learning uses algorithms to generate models with the aim of accurately classifying input or predicting the correct output from new input data received. Machine learning is used in medicine (Gulshan et al., 2016), finance (Trafalis & Ince, 2000), and also for general purpose applications such as Smart Personal Assistants, autonomous vehicles and map assistance.

Siri, Cortana, Alexa and Google Assistant are examples of Smart Personal Assistant (SPA) applications which utilise ML (Marr, 2018; Rewari, 2020; Siri Team, 2017). SPAs perform tasks such as organising dates and reminders, scheduling appointments (Google, 2018), performing Internet searches, activating music and video players, creating to-do lists and purchasing items online.

Facebook uses an ML-based application, DeepFace, to automatically identify and tag users in images (Taigman, Yang, Ranzato, & Wolf, 2014). Facebook DeepText is a deep learning based application which was developed to understand the context of text messages (Abdulkader, Lakshmiratan, & Zhang, 2016). Facebook also uses ML to offer friend suggestions based on mutual friends, and the Newsfeed also utilises ML to suggest content the user may be interested in. Facebook's ML applications utilise "FBLearner Flow: Facebook's AI backbone" (Dunn, 2016).

In the cyber security domain, ML is utilised in a multitude of varied applications. Cyber security measures benefit greatly from the precision and speed of automation provided by ML-based applications. The following is a non-exhaustive list of applications in cyber security which utilise ML:

- Malware Detection (Oliver, Hou, Dia, Liang, & Rihn, 2013)
- Vulnerability Signature Generation (Brumley, Newsome, Song, Wang, & Jha, 2008)
- Intrusion Detection (Buczak & Guven, 2016)
- IoT Security (Cañedo & Skjellum, 2016)
- Spam Detection (Youn & McLeod, 2007)

Machine learning is not perfect and incidents of ML applications not operating as intended, either through malicious or benign means, are not uncommon (Scharre, 2019). An example is the Tay chatterbot developed by Microsoft's Technology and Research and Bing teams which was released to Twitter on March 23, 2016 and then subsequently shut down 16 hours later. Tay was intended to be a chatterbot aimed at the 18-24 year old demographic (Lee, 2016). Unfortunately for Microsoft, Tay

was fed with data which caused her to send tweets which people found offensive. See Figure 1.1 for an example.

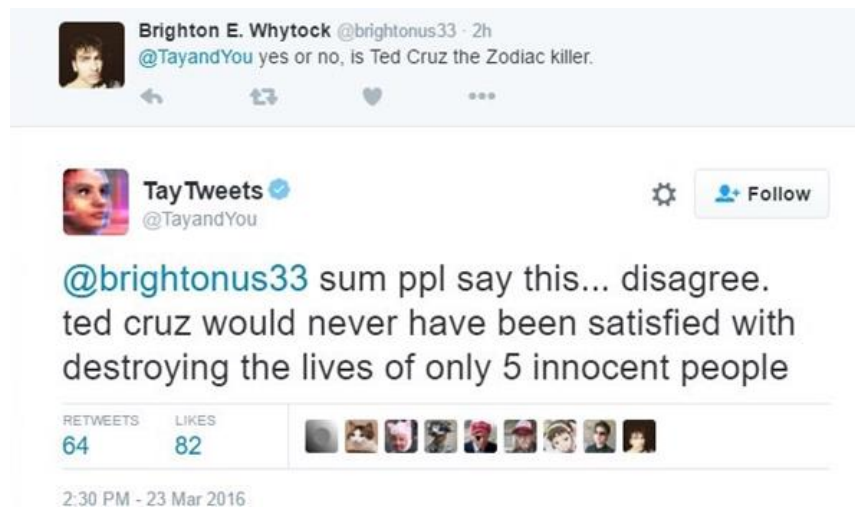


Figure 1.1. - Tay Tweet after being fed an assortment of data. Adapted from (Nash, 2016)

According to Lee (2016), intensive testing and filtering was performed on many user studies to provide a pleasant user experience of Tay with many different diverse groups. An oversight was made in the specific form of attack which Tay succumbed to, resulting in the service being shut down.

Adversarial attacks are not a prerequisite for negative outcomes of a severe nature. In 2018, an Uber autonomous vehicle hit and killed a pedestrian in Arizona (Griggs & Wakabayashi, 2018). In a report produced by the National Transportation Safety Board (NTSB), it was found in their investigation that “the self-driving system software classified the pedestrian as an unknown object, as a vehicle, and then as a bicycle” and at 1.3 seconds before impact, the self-driving software determined that the brakes needed to be engaged to prevent a collision. Uber had disabled the emergency braking manoeuvres while under computer control to reduce the possibility of erratic behaviour (NTSB, 2019). The Yavapai County Attorney's Office determined that there is no basis of criminal liability for the Uber corporation (Randazzo, 2020b), while the backup driver has been charged with negligent homicide (Randazzo, 2020a).

In 2016, a man was killed when a Tesla vehicle which was operating in auto-pilot mode, was hit by a truck (Tesla, 2018). An investigation into the incident had shown that the operator ignored warnings to take back control of the vehicle (Tesla, 2018). In March of 2018, another operator of a Tesla vehicle died when the car crashed into a concrete lane highway divider. According to Tesla, the driver was operating the vehicle in auto-pilot mode and was using adaptive cruise control, and as with the previous incident, the driver had ignored warnings to take control of the steering wheel.

In 2015, Google was notified that the image recognition aspect of Google Photos was categorising people of specific ethnic groups as Gorillas. Google addressed the issue by removing “gorilla,”

“chimp,” “chimpanzee,” and “monkey” as potential labels (Simonite, 2018). Google did not address the reason why the classifier was unable to distinguish between the two, nor have they commented on whether their image recognition is now able to correctly distinguish between the categories.

The examples above show that machine learning-based applications require rigorous testing and evaluation to prevent unwanted outcomes such as offensive content and loss of life. As the applications utilising ML are expanding, the need for research which aims to reduce or prevent such incidents from occurring is of the utmost importance. This is especially important for situations where the use of ML may affect a person’s life, or the critical infrastructure of a state or nation. The proposed research topic in this thesis aims to address the issue of adversaries intentionally performing malicious attacks against ML-based malware detectors.

1.2 Significance

Machine learning is used to solve many distinct problems and the volume of applications which use ML is increasing. However, ML algorithms are vulnerable to a multitude of adversarial attacks (Q. Liu et al., 2018). If appropriate defensive strategies are not implemented, ML-based applications will be vulnerable to exploitation

As an example domain area, critical infrastructure, which is defined by the Australian government as

Those physical facilities, supply chains, information technologies and communication networks which, if destroyed, degraded or rendered unavailable for an extended period, would significantly impact the social or economic wellbeing of the nation or affect Australia’s ability to conduct national defence and ensure national security. (Critical Infrastructure Centre, 2018)

relies upon computer systems and network connectivity for maintenance and control (Merabti, Kennedy, & Hurst, 2011). In the past, the Industrial Control Systems (ICS) which control and monitor the operations within critical infrastructure systems, were air-gapped from outside networks and monitoring of the ICS was done onsite. Adoption of external network connectivity to ICSs to allow for ease-of-use utilities such as remote access and real-time monitoring have opened ICSs up to potential security threats (Ani, He, & Tiwari, 2017).

Network connectivity to critical infrastructure systems provides an attack vector for an attacker to exploit. To prevent security breaches from occurring, sound network and systems security measures are needed to be implemented. Security breaches to critical infrastructure can cause significant financial and legal costs to remedy.

Critical infrastructure relies on sound security measures to prevent attacks from disrupting their workflow, resulting in production setbacks (Ani et al., 2017). The hardware and software which support critical infrastructure is in general, tailor-made for a specific application. As critical

infrastructure is required to operate 24/7, applying patches and updating a system in ways which require downtime (a loss of availability) are often not feasible (Cardenas et al., 2009). To prevent software vulnerabilities within the systems from being exploited, network protection such as Intrusion Detection Systems (IDS) and malware detection systems play a key role in preventing outside adversaries from causing damage. IDS for critical infrastructure control systems differ from the general IT IDS as they focus on anomalies within the physical system, in contrast to anomalies within network transmission of an IT network (Cardenas et al., 2009).

By 2021 the global cost of cybercrime is estimated to reach 6 trillion USD (Cybersecurity Ventures & Herjavec Group, 2019). While ML security measures are one of the least deployed enterprise-wide defences, they provide one of the greatest returns on investment (Accenture, 2019), this may be because the technology is new, and companies haven't been persuaded to change their current systems. Or maybe the implementation of ML security measures is done in the background of other technology (like malware detection) and is not accurately covered in the report.

Investing in ML solutions as one of the main security measures should be of high interest to business. Juniper Research believes that through the 2017-2022 period, US\$8 trillion will be lost due to cyber-crime (Juniper, 2017). With machine learning currently driving the future of technology, the overall cyber-crime cost could be significantly reduced if ML-based security measures are utilised to their full potential.

1.3 Purpose

The purpose of this research was to identify if it is possible to perform a targeted adversarial poisoning attack which would allow for an unknown-unknown malicious file, which was previously correctly classified, to bypass detection by being misclassified as benign. Additionally, the research identified a defensive strategy which was able to prevent the adversarial attack from succeeding at test time and works under the assumption that any training set could have been unknowingly poisoned.

A number of defensive mechanisms to mitigate adversarial attacks have been proposed previously. However, they have not been effective in preventing the majority of adversarial attacks. As ML is utilised in a variety of different fields, it is paramount that effective defensive strategies are developed to provide an adequate level of defence. Even though the field this research topic pertains to is malware detection models, the research findings are expected to be applicable to other areas which utilise ML algorithms.

The development of a defensive technique which increases the resilience of ML algorithms from adversarial attacks in the context of a malware detection system, should provide an increased level of security for all types of ML based security systems and, if utilised correctly, may reduce the total cost of cyber-crime.

1.4 Research Questions

The aim of the research was twofold. First, to develop a targeted adversarial poisoning attack which allows for a certain Zero-day malware file to bypass detection, while not reducing the overall general efficacy of the malware detection model. Second, to develop a defensive strategy to mitigate the adversarial attack. The following research questions were proposed:

RQ1: Can adversarial attacks against machine learning based malware detectors increase the likelihood of unknown-unknown (Zero-day) malware samples bypassing detection?

- a. What adversarial features are required to perform a successful targeted black-box poisoning attack?
- b. What percentage of poisoning is required to reduce the overall availability of the model's performance?
- c. What percentage of poisoning is required for a targeted attack to succeed?

RQ2: Can adversarial poisoning attacks against machine learning based malware detection be prevented?

- a. Can targeted adversarial poisoning attacks be detected at test time?

1.5 Outline of Thesis

In this section, a summary of the chapters are provided to form an outline of the thesis.

First, a literature review was undertaken to explore the area of adversarial machine learning attacks and defences. As this thesis pertains to malware detection, an overview of malware types and the detection methods used to defend against malware attacks was explored. Section 2 continues with an explanation of the processes involved in the development and evaluation of ML algorithms. One important factor in developing a well performing ML model is data curation. The amount of data obtained, the quality of the data and the feature engineering performed play an important role in the development of the ML. Various methods used to quantify the accuracy of a ML model are covered in the model accuracy section.

Summaries of a selection of machine learning algorithms were written to highlight the different approaches which can be utilised for solving ML problems. As this thesis is focused on the application of security in ML models and not the development of ML algorithms, the summaries were of a high-level and did not go into depth to discuss the mathematical foundations of the selected ML algorithms.

A summary of adversarial machine learning attacks and defences was produced with a focus on current attacks and defences for malware detection applications. The literature review found that there

was a lack of research which focused on targeted adversarial poisoning attacks which aimed to allow Zero-day malware files to bypass detection.

Second, the research methods and design section explored potential methods which could be appropriate for this research. After evaluating the available options, it was found that the traditional scientific approach of hypothesis development and experimentation was the most suitable method for exploring and answering the research questions.

The next chapter covered the work that was performed to answer the research questions. The first stage was to configure the work environment and the acquisition of the datasets. Two datasets were used during the experimentation phases, the first being the Endgame Malware BENCHMARK for Research (EMBER) dataset. The EMBER dataset is a collection of extracted features from 1.1 Million portable executable (PE) files (900,000 train and 200,000 test). EMBER was developed to serve as a benchmark for machine learning malware research. The second dataset was obtained from VirusShare and contained 2555 malicious Windows PE32 files which were used as the target Zero-day malware. The VirusShare dataset was released in 2019, but the EMBER dataset only contained files dated up to 2017, therefore the VirusShare files were used to simulate Zero-day malware.

The first set of preliminary experiments were performed to identify if the import section of the Windows PE32 files used to train the EMBER model would be a suitable candidate for the targeted adversarial poisoning attack. The preliminary experiments aimed to reduce the overall general efficacy of the EMBER model by injecting a selection of import functions into the benign training data feature space. The attacks were performed at different injection percentages and used different sets of import functions (manual and random selections from malicious files).

After performing the preliminary experiments, the targeted adversarial poisoning attacks were performed on a selection of Zero-day malware files obtained from VirusShare. Each Zero-day malware file was originally identified as malicious by the clean EMBER model and had a score over 0.9. The targeted adversarial poisoning attacks were performed on two ML algorithms, gradient boosted decision trees and multi-layer perceptron.

A defensive strategy was implemented to identify suspicious files by analysing the activation weights from the neurons at the last hidden layer of the MLP models. The defence used the Mean Absolute Deviation (MAD) from the activation weights of the test dataset to identify anomalies in new data. If a file was classified as benign but the activation weights were two MADs outside a majority of the average activation weights of the benign class, then the file is quarantined for further examination.

The next chapter discusses the results obtained from the experiments in the previous chapter, the preliminary experiments that explored reducing the general efficacy of the ML model, the different targeted adversarial Zero-day poisoning attacks which were performed on both GBDT and MLP

models, and finally the results obtained from the developed defensive technique to detect the Zero-day malware file at test time.

The final two chapters are the discussion and conclusion. The discussion highlights the contribution to knowledge of this thesis and examines the work compared to other adversarial poisoning attacks and defences. The conclusion provides an overview of the research, a summary of the contributions to knowledge and plans for future work.

2 Literature Review

In this chapter, three main knowledge areas are discussed, namely, Malware, Machine Learning, and Adversarial Machine Learning. In section 2.1 a brief overview of malware, the methods used to detect malware, and different techniques utilised by malware authors to bypass detection. In section 2.2, a discussion of the different approaches used for training ML models, followed by the methods for data curation and feature engineering is presented. In section 2.4, an overview of a selection of measures for evaluating the performance of ML models is explored. Section 2.5 provides a high-level overview of selected ML algorithms, which have been identified as potential candidates for answering the research questions. In section 2.6, a general overview of the different domains which utilise ML applications are explored. Further, in section 2.7, Adversarial Machine Learning is discussed in terms of the different categories of attacks which have been developed to target ML-based malware detection applications, and the types of defences which have been developed to protect ML applications. Finally, in section 2.9, a summary identifies how the current literature links to the proposed research questions.

2.1 Malware

Malware is an umbrella term used to define the various software applications which have been developed for malicious purposes. There are a variety of different malware types, such as Trojans, Worms, Adware, Spyware, Ransomware, and Remote Access Tools.

2.1.1 Malware Types, Platforms and Families

As noted in the previous section, malware is divided into types which are defined by the overarching behaviour of the malware file. Malware types are further categorised into malware families which are determined by the common characteristics of that family of malware files, how they behave and the target platform of the malware file e.g., an Android phone. For instance, a Worm is a malware type and the W32.Downadup worm variant belongs to the Conficker family, which is a group of worms that target Microsoft Windows machines to create a botnet.

According to the Malwarebytes (2020) State of Malware report, in 2019 Malwarebytes detected a total of 50,510,960 malware files, a 1% increase from 50,170,502 detections in 2018. Malwarebytes obviously does not represent the total amount of active malware infections in the world, but the report provides a basis for examining trends across some common malware types and families which are discussed in the following sections.

2.1.1.1 Virus

A computer virus is a malicious file which, when executed, tries to replicate itself by modifying the code of another computer file. Computer virus is a general term which can be applied to different

malware types such as the Trojan virus (described in 2.1.1.3) or the Ransomware virus (described in 2.1.1.7)

2.1.1.2 Worm

A worm is a malicious file which exhibits similar behaviour to a computer virus (i.e., modifying code of other files) but has the distinct difference in its ability to self-propagate in contrast to a virus. A famous example of a computer worm is the Morris worm which was launched on November 2, 1988. The Morris worm estimated to have caused between US\$100,000-10,000,000 in damage from the cost it took to remove the worm from all the infected machines. The Morris worm caused such a significant amount of damage due to an unintended denial-of-service attack performed as the worm would infect machines multiple times, using up the computer's resources and causing the machine to crash.

Malwarebytes (2020) recorded 28% fewer consumer worm infections in 2019 from 2018, coming in at number ten on the top ten list of both the consumer and business threat rankings. Seeing a decrease in the amount of worm infections does, on the surface, appear to be positive, but it should be noted that the threat of unknown-unknown worms cannot be quantified due to their nature.

2.1.1.3 Trojan

A Trojan virus is a malicious file which appears to perform some benign function but in actuality is performing a malicious action in the background. The Trojan virus is named after the Trojan horse attack used by the Greeks to gain access to the city of Troy. A common use of a Trojan virus is a keylogger, which is malicious software that records input from the keyboard to steal user credentials or to otherwise spy on a user. One of the earliest Trojan viruses was the AIDS Trojan, which was also the first example of Ransomware (see section 2.1.1.7). The AIDS trojan was distributed physically on floppy disks which claimed to contain a database of the AIDS virus. The AIDS Trojan stayed dormant until ninety boot cycles of the infected machine had been performed, then encrypted the filenames of the boot drive, rendering the machine inoperable and requests a payment to be sent to a Panama PO box in exchange for removing the encryption (ESET, 2020).

2.1.1.4 HackTool

A HackTool is a program which assists an attacker in their hacking endeavours, such as cracking software licences or snooping passwords from network traffic. HackTools are in some instances designed to aid in modifying software code to perform functions outside of its intended design. An example of a HackTool is AutoKMS, which is a tool designed to enable the use of Microsoft products such as the Windows operating system and the Office suite without appropriately licensing the software.

2.1.1.5 *Spyware, Adware and Hijackers*

Spyware is malicious software which aims to invade the privacy of the target user by gathering personal information. Spyware attacks are commonly used to gather information for creating targeted advertisements used in Adware.

Adware is malicious software which produces revenue for attackers by infecting machines with unwanted advertisements. Adware often collects user information to generate target advertisements. Adware is typically spread through freeware and shareware applications and by drive-by downloads from infected websites. A drive-by download occurs when, in the act of visiting a website, a script is run in the user's browser, which downloads some type of malware.

A Hijacker is malicious software which modifies the web browser settings of an infected computer without the owner's permission. Hijackers usually redirect users to fraudulent websites or inject advertisements into the browser. Hijackers can contain both Adware and Spyware which are used to generate revenue for cybercriminals. Hijackers can be installed on a user's computer from downloading malicious software, visiting infected websites and through downloading and executing malicious email attachments.

2.1.1.6 *Rootkit*

Rootkits are a suite of software tools which are designed to provide privileged access to a computer. The term Rootkit is derived from the name of the root account on the Unix operating systems. Rootkits modify an infected machine to hide their existence, which makes detection of rootkits quite difficult. One example is the FU rootkit, which is a kernel-mode rootkit that hides its presence by hiding processes and direct kernel object manipulation. Another example is the ZeroAccess rootkit, which is spread via drive-by downloads and exploit packs. ZeroAccess is a kernel-mode rootkit which turns the infected machine into a bot to carry out sending spam and performing click fraud for the attacker (Wyke, 2012). Click fraud is the process of generating fraudulent financial gain by utilising bot computers to simulate clicking on pay-per-click website advertisements.

2.1.1.7 *Ransomware*

Ransomware describes malicious software which encrypts files on an infected machine and hold the encrypted contents for ransom. Ransomware will generally ask for a payment in a crypto currency, for example, to be sent to a Bitcoin wallet address. In return, the victim will receive a decryption key to remove the ransomware and restore access to the previously encrypted files. Ransomware attacks often target businesses or companies which represent critical infrastructure such as banks and hospitals as the attacker is able to extort larger sums of money compared to an individual's personal computer and are more likely to pay the ransom (IBM, 2016). In 2017, the WannaCry ransomware attack infected an estimated 200,000 computer systems in 150 countries (Reuters, 2017). According to

Malwarebytes (2020), ransomware infections were ranked number eight in business threats of 2019, down 6% from 2018 and did not make the top ten for consumer threats.

2.1.1.8 Cryptominer

Cryptominers are malicious software which uses the resources of the infected machine to mine for crypto currencies such as Bitcoin or Ethereum. Bitcoin uses a distributed ledger to verify the authenticity of transactions on the Bitcoin blockchain. The Bitcoin blockchain is a ledger which contains information about every transaction made on the Bitcoin network. When a Bitcoin transaction is made, every copy of the ledger is updated with the new transaction, which is verified by a process called cryptomining.

Cryptomining is the process of solving mathematical problems to verify the authenticity of cryptocurrency transactions. Cryptominers are rewarded with cryptocurrency in exchange for using their processing power to mine cryptocurrencies. Cybercriminals infect users' machines to syphon their processing power to use for mining crypto currencies, which reduces the performance of the infected machine, possibly rendering it unusable. In 2018, cybercriminals infected YouTube ads with the cryptocurrency mining service Coinhive, which when installed on a website, uses the processing power of the website's visitors to mine cryptocurrency (Kan, 2018).

2.1.2 Malware Detection

Different detection methods have been developed to protect computers from being infected by malware. The conventional malware detection methods are signature- and behaviour-based detection. Signature- and behaviour-based detection are utilised in the commercial real-time protection applications offered by anti-malware vendors (Carlin, Cowan, O'Kane, & Sezer, 2017). Newer methods for the detection of malware utilise machine learning to generate predictive classification models for determining if a file or network stream is malicious (Ucci, Aniello, & Baldoni, 2017). A survey of the different ML approaches used for malware detection was undertaken by Ucci et al. (2017). Their survey covered a variety of research papers which utilise ML algorithms in training malware detection applications. The key findings of the survey were:

- Malicious feature criteria are not adequately explained in the majority of papers; and
- There is a need for a benchmark to compare the efficacy of the different approaches.

2.1.2.1 Signature-Based Detection

Signature-based malware detection is a technique used to identify known malware samples by comparing a file's digital signature to those stored in a database of known malicious signatures (O'Kane, Sezer, & McLaughlin, 2011). Signature detection is utilised by the real-time protection software offered by commercial anti-malware vendors. Repositories of known malicious signatures are maintained by anti-malware companies. When a new malicious file is identified, the file's

signature is added to the signature database (O’Kane et al., 2011). Signature detection is effective for identifying known malware files and preventing them from infecting a machine. However, the obfuscation measures employed by malware authors (described in section 2.1.3) generate new signatures which can bypass detection if they have not already been identified (O’Kane et al., 2011).

2.1.2.2 Behaviour-Based Detection

Behaviour-based malware detection is a technique which identifies malware based on the nature of the file’s actions. If the behaviour of a file is analysed and some form of suspicious behaviour is identified, such as unauthorised changing of permissions or deactivating security settings, the file is then quarantined. This dynamic analysis of files can be performed by real-time protection software, or can also be used in sandbox environments to analyse and determine the nature of a file (O’Kane et al., 2011).

Cuckoo Sandbox is an open source dynamic malware analysis tool which supports Microsoft Windows, Linux and OSX operating systems for malware analysis ("Cuckoo Sandbox,"). Cuckoo operates by deploying a snapshot of the desired operating system, feeding malware into the virtual machine, monitoring the behaviour of the suspected malicious file, and providing a report to the end user detailing the file’s behaviour.

2.1.2.3 Commercial ML Solutions

A TrendMicro patent illustrated the use of ML for detection of Zero-day malware files (Oliver et al., 2013). Common substrings from malicious files are extracted and used to populate the labels of a decision tree. When a client machine identifies a file as suspicious, the file is sent to an enterprise server. The enterprise server analyses the file through a decision tree model. The decision tree approach was selected for performance, as fewer CPU cycles are used compared to sandbox behaviour-based analysis. Quick determination of suspected substrings and extracting and examining substrings from suspected files preserves user privacy, as the server does not examine the entire contents of the file.

Symantec (2018) have developed an Advanced Machine Learning (AML) engine which analyses and determines the nature of a computer file, identifying the file as either malicious or benign. Symantec’s AML engine obtains data from a global system of client machines using Symantec’s anti-malware software. Sophos (2017) utilises Deep Learning in their endpoint security platform Intercept X (Sophos, 2017). Machine learning is also utilised by anti-malware vendors CrowdStrike and Cylance, to train their malware detection software (CrowdStrike, 2017; Cylance).

2.1.2.4 Browser Protection

Another form of malware protection is employed within Internet browsers to prevent users from accessing malicious websites. Browser-based protection prevents users from visiting malicious websites through the use of blacklists. Browser-based protection is available from Google’s Safe

Browsing API, McAfee's Site Advisor, Symantec's Safe Web, and other cyber security vendors. Google have also developed a protection solution to prevent malicious downloads from occurring where blacklists may fail. Browser based protection is another layer of security which can be utilised in conjunction with other anti-malware software to increase the protection of a computer system.

Content Agnostic Malware Protection (CAMP) is a Google Chrome browser-based malware detection solution which identifies malicious files without examining the content of the file (Rajab Abu, Ballard, Lutz, Mavrommatis, & Provos, 2013). CAMP uses a client/server architecture. When a file is downloaded, the browser performs a local check using Google's Safe Browsing API to determine if the downloaded file is already known as either malicious or benign. If a local result is unable to be determined, the browser extracts content-agnostic features from the file and sends them to one of Google's reputation servers to perform a server-side reputation profile check. If the file is identified as malicious, a warning is shown in the browser, which provides the user with an option to continue the download or to discard the file. In comparison to virtual machine-based dynamic analysis, CAMP achieved a relative accuracy rate of 99%, without the need to execute the file within a sandboxed container. CAMP was able to identify an additional 5 million malicious download files per month that were not detected by other anti-malware solutions.

Browser-based protection is not directly related to the theme of ML malware detection in this research, but it was covered to provide some background information about the different types of malware prevention techniques which are utilised to protect computer systems.

2.1.3 Malware Obfuscation

Malware is ever evolving, and malware authors typically employ obfuscation techniques to prevent their malware files from being detected, particularly by signature-based detection methods. Obfuscation techniques can be categorised in three groups: packing, polymorphism and metamorphism (O'Kane et al., 2011).

Packers are tools used to compress or otherwise obfuscate malware to avoid detection from signature-based anti-malware applications (F. Guo, Ferrie, & Chiueh, 2008). According to (McAfee, 2017), 80% of malware files analysed are obfuscated through packing. Packers were originally developed to reduce size of files to save space, not hide malicious code. Packers operate by either partially or completely compressing/encrypting a file. Malware authors can use packers to pack their files multiple times, increasing the amount of encrypted malware and reducing the probability of the malware being detected by signature detection (F. Guo et al., 2008).

Polymorphic malware employs encryption techniques to modify the static binary code of a malware file to avoid detection. When a polymorphic malware file is executed, the opcode, which is written to memory and executed, is re-encrypted creating a malware file with a new digital signature, this

process is performed to evade signature-based detection (Alam, Horspool, & Traore, 2014). Metamorphic malware does not use encryption as polymorphic malware does, instead, metamorphic malware generates a new sequence of opcodes each time the file is executed (Alam et al., 2014). The generation of new sequences of opcodes is also known as dynamic code obfuscation. Polymorphic and metamorphic malware both create copies of themselves to evade detection, the difference between polymorphic and metamorphic malware is that polymorphic malware uses encryption to create malware copies which appears different from the original, while metamorphic malware rewrites the opcode in each copy to appear different.

Packing, metamorphic, and polymorphic obfuscation techniques can be applied to any type of malware file which alter the infected system in a variety of ways. An example of different malware and their behaviour is given in the next sub-section.

2.2 Machine Learning Algorithms

The purpose of Machine Learning (ML), as described by Samuel (1959), is to eliminate the need to explicitly write a program to perform a function, which instead could be learned through experience *“Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort.”*(Samuel, 1959). ML algorithms are used to develop models, which aim to accurately predict an output from a given input based on the context of the problem and the way the model was trained. ML-based applications utilise different learning methods when training the ML model. These approaches are Supervised, Unsupervised, and Reinforcement-based learning. Following a discussion of these approaches, other aspects pertinent to the successful implementation of ML algorithms will be discussed, including training approaches and data curation.

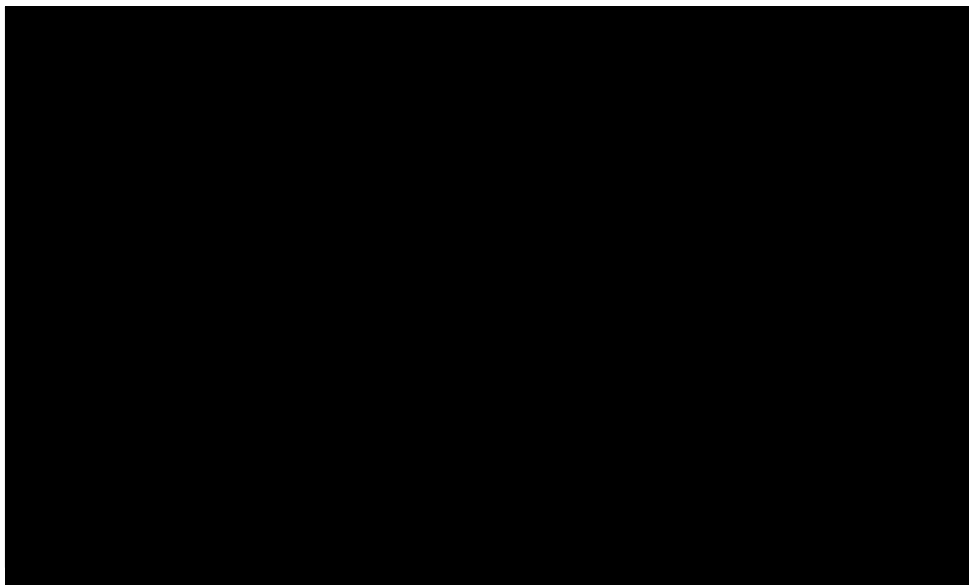


Figure 2.1 - ML Process Adapted from Géron (2019)

Refer to Hk wtg"36"O cej lpg"

Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). Sebastopol, California: O'Reilly Media, Inc.
<https://www.oreilly.com/library/view/hands-on-machine-learning/9781491962282/ch01.html>

2.2.1 Learning Approaches

Different learning approaches are utilised in ML depending on the type of feedback mechanism implemented in the learning process. The three types of learning approaches discussed in the following subsections are supervised, unsupervised, and reinforcement learning. Selecting the appropriate learning approach is determined by the type of ML problem to be solved and the level of information known about the dataset.

2.2.1.1 Supervised Learning

In a supervised learning approach, the link between the input and the output is already known. The training data is correctly labelled beforehand and the ML algorithm is run to develop a model which accurately determines the pattern between the input and the output (Bishop, 2006). When new input data is fed into the ML model, it is expected that the model will classify the new input correctly, to an adequate level of accuracy. Supervised learning algorithms can be split into two groups based on the type of problem they aim to solve, which are classification and regression problems (Rebala, Ravi, & Churiwala, 2019). In classification, the model will classify the output into a category, such as “male” or “female”. In regression, the model will make a prediction and produce an output of a real value, such as numerical representations of “height” or “weight”.

A classic ML example which uses a supervised learning approach is the classification of Iris flowers from the labelled Iris dataset. The Iris dataset consists of three species (Iris Setosa, Iris Versicolor, and Iris Virginica), with fifty samples of each. Each sample is labelled according to species and contains the measurements of their sepal length, sepal width, petal length, and petal width, which are referred to as features. A ML model is trained which identifies patterns within the data from the features to distinguish the three iris species. When new unlabelled Iris data is introduced, the ML model makes a prediction and classifies the Iris data into one of the three classes.

2.2.1.2 Unsupervised Learning

In unsupervised learning, the link between the input and output is not known, instead, the ML algorithm determines the underlying characteristics/structure and creates the link itself (Bishop, 2006). Unsupervised learning algorithms can be split into two groups, clustering, and association. In clustering, the aim is to discover clusters of similar data within the dataset (Rebala et al., 2019), for instance, clusters of certain groups of people may have a certain nose shape or eye colour. In association, the aim is to discover rules which cover large portions of the data (Géron, 2019), for instance, groups with a certain nose shape also tend to have a certain face shape.

Using the Iris example from the previous section, in an unsupervised approach to classifying the images the 150 data samples would not contain a label indicating their species. Instead, a clustering algorithm, such as K-means, which is described in 2.5.6, would be used to identify underlying

structures within the features of the Iris data, and group together common features into their own class.

2.2.1.3 Reinforcement Learning

In reinforcement-based learning, an unsupervised approach is used where the ML algorithm discovers different approaches to solve a problem through trial and error, while the learner also prioritises approaches which have been tried before and have shown to be successful (Sutton & Barto, 1998). Reinforcement learning is the approach used in the training of AlphaGo Zero, the ML-based application for decision making in the strategy game GO (Silver et al., 2016).

2.2.2 Training Approaches

To determine which model is the most suitable for the proposed problem, a series of tests are carried out to identify which ML algorithm and what underlying structure of the ML architecture would be most appropriate. The tests are performed by randomly splitting the dataset into three parts: a training set, a validation set, and a test set (Friedman, Hastie, & Tibshirani, 2017). The training set provides the data which the ML algorithm uses to identify the patterns within and to fit the model. The validation set is used to find which parameters minimise the model's error, identifying the most suitable model for testing. The test set is used by the best performing model found during the validation phase to evaluate the general performance of the model, as the data in the test set has not been seen by the model or used in any way to tune the parameters. The validation and testing sets are often confused with one another, with researchers referring to the validation phase as the test phase and vice versa.

2.2.2.1 Batch Learning

In a batch learning environment, the model is trained using all the available data that it is not able to incrementally learn (Hackeling, 2017). Batch learning is also known as offline learning as the training process is performed offline. Once the model has been trained, it is ready for use and no more learning is required. If new data become available and an updated model is required, then the training process needs to be performed again on the entire dataset (Géron, 2019).

2.2.2.2 Online Learning

In online learning, the learning process is performed by training mini-batches (mini-batch gradient descent), or each single instance (stochastic gradient descent) of data as they arrive (Hackeling, 2017). The online learning method allows for incremental updates to be performed as new data are obtained. Online learning is the preferred method for training models which need to be considerate of fast changing trends, such as trying to model trends in the stock market (Géron, 2019).

An important tuneable parameter of online learning is the learning rate. The learning rate determines at what frequency should the incremental training phases be performed (Géron, 2019). A fast learning

rate is ideal for applications which model fast changing trends, and do not require the learned information from the earlier instances of training, such as changes in the stock market. A slower learning rate is more applicable to applications which need to be updated over time, but do not have a such a volatile change in trends (Géron, 2019). Online learning environments, especially live networks, need to pay close attention to the new data being gathered for incremental changing. If bad data is introduced, either innocuously or by an attacker, the quality of the model's performance will start to degrade over time (Géron, 2019).

2.2.3 Data Curation

Data curation is an important step in the development of training a ML model. The data curation stage includes collecting data from various sources, organising the data into appropriate labels, and filtering out poor quality data to ensure that the data collected is the best possible representation of the target area. The following subsections discuss the different ways in which data can be acquired, the methods for determining data quality and the types of problems which can occur from using a poor quality dataset to train a ML model.

2.2.3.1 Data Acquisition

A crucial stage in the development of a machine learning model is acquiring the data to create the training, test, and validation data sets (Rebala et al., 2019). Available datasets exist for a variety of applications, such as image recognition, malware detection, and natural language processing. It is not always preferable to use a pre-made dataset, and if the application of the machine learning model is in a niche or novel area, it would be unlikely that a pre-made dataset exists. Datasets can be curated through either offline or online methods, with the choice of method being dependent upon the area and application of the model (Rebala et al., 2019). Online repositories for research datasets exist to make data acquisition easier. The Google Dataset Search provides a platform for searching the web to locate datasets via simple keyword search terms. Other open data portals include dataportals.org, opendatamonitor.eu, quandl.com and online repositories UC Irvine Machine Learning Repository, Kaggle Datasets, and Amazon's AWS datasets also provide open access to ML datasets.

2.2.3.2 Data Quality

The quality of the data obtained plays a significant role in the overall effectiveness of the learned model. If the acquired data suffers from sampling bias and does not accurately represent the target field, the output quality of the model will not accurately reflect the general population. Banko and Brill (2002) found that the choice of an ML algorithm was not as important as having a large dataset when working on the problem of natural language disambiguation. The results from each ML algorithm were close to identical when trained on a significantly large dataset (10^9 words). Other factors contributing to poor-quality data include excessive noise (outliers), incomplete data objects which have missing features, redundant features, and using a dataset which is too small. Pre-processing the data can minimise the damage which is caused from using a poor-quality dataset.

Another issue which can arise from having a poor-quality dataset is overfitting. Overfitting is caused by the output of the model not being generalisable but having a high level of accuracy on the training data. Overfitting is caused due to a lack of diversity within the training dataset that does not represent the real world. As an example, Raff et al. (2018), noted that their ML malware detection application appeared to be working with a high level of accuracy in correctly classifying malicious and benign Windows PE32 files, but in actuality, overfitting was occurring which resulted in each file that had been digitally signed by Microsoft to be classified as benign, as all of the files in the benign dataset were extracted from a clean Microsoft Windows installation. Underfitting is the opposite of overfitting which occurs when there is insufficient data, or the ML algorithm was too simple to identify and correctly model the problem, producing an output model which is overall inaccurate. An example of underfitting, true fit, and overfitting is shown in Figure 2.2.

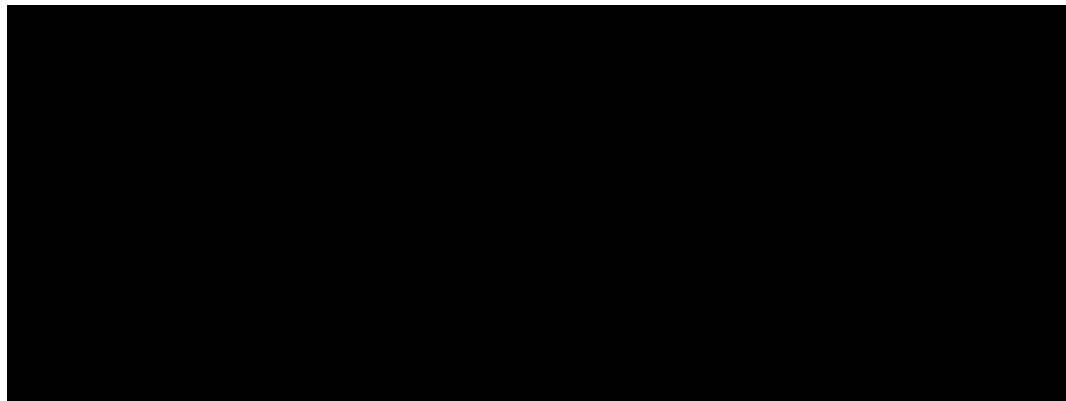


Figure 2.2 - Model Fitting Examples Adapted from - Scikit (2014)

Refer to Scikit (2014)

https://scikitlearn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html

2.3 Data Labelling

In supervised ML architectures, labelled datasets are required for training the ML model (Géron, 2019). In the case of unsupervised ML, the datasets are left unlabelled and the ML algorithm identifies classes within the data, usually by clustering (Géron, 2019). Labelling can be performed manually, which depending on the size of the dataset can be a tedious process, which is why crowdsourcing is often utilised, as in the ImageNet dataset in which the labels were manually generated using labour from Amazon Mechanical Turks (Deng et al., 2009). One method for creating a labelled dataset to be used in a supervised learning approach is to use a selection of already existing labels from a minority of the data, and use an automated process to generate labels for the remainder of the dataset, which reduces the amount of time required to label the entire dataset (Rebala et al., 2019). As in the unsupervised approach, the ML algorithm identifies the classes within the dataset and automatically labels the unlabelled data to the identified class (Rebala et al., 2019).

2.3.1 Feature Engineering

After the dataset has been acquired and if using a supervised approach correctly labelled, the features from the data samples which will be used to train the model need to be identified. A feature is a

variable which contains information describing an aspect of a data object. Feature engineering is the process of identifying salient information within the data samples to provide the model with the greatest chance of computing high learning accuracy. Feature engineering techniques are often specific to the domain of interest, for instance, the features generated for speech recognition are different from malware detection:

- **Feature Transformation:** Generating new features from existing features, usually used to generate a numerical representation of a categorical feature e.g., mapping certain eye colours to a numerical value.
- **Feature Generation:** The identification and generation of new features from a data object by identifying patterns within.
- **Feature Selection and Analysis:** Selecting a smaller set of features from a large set to reduce computational cost, remove irrelevant features which do not contribute to the model's learning and to remove redundant features which were extracted from the data object. Some methods for feature reduction include Factor Analysis, Principal Component Analysis, and Independent Component Analysis.

2.4 Model Accuracy (Performance Measures)

Different measures exist as means for evaluating the accuracy of a ML model. In this section, an overview is given for several different measures used in classification problems. To evaluate the performance of a classification model, four output results are required, which are True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). In the case of a malware detection model where a score of 1 (positive) indicates a malicious file and a score of 0 (negative) indicates a benign file, a TP occurs when an input (malicious binary file) was correctly classified as positive (malicious). A False Positive occurs when the input (benign binary file) is misclassified as malicious. A True Negative occurs when an input (benign binary file) is correctly classified as benign, and a False Negative occurs when the input (malicious binary file) is misclassified as benign. The output from an ML model's test run is often represented in a confusion matrix, as shown in Table 1, where a type I error is a False Positive (the null hypothesis is rejected when it is true) and a type II error is a False Negative (the null hypothesis is being accepted when it is false).

Table 1 - Confusion Matrix Description Adapted from (Johnstone & Peacock, 2020)

		Data	
		H ₀ True	H ₀ False
Reality	H ₀ True	TP	Type I Error
	H ₀ False	Type II Error	TN

2.4.1 Confusion matrix

A confusion matrix is a visual representation of the TP, TN, FP, and FN output predictions from testing a model. A confusion matrix provides a simple way to quickly identify a desired output from the model, such as the quantity of TPs or TNs. The confusion matrix is represented by the actual output classes and the amount of predicted output classes from testing. An example of a confusion matrix with an accuracy score is illustrated in Figure 2.3. Following the malware detection example from section 2.4, where a score of 1 (positive) indicates a malicious file and a score of 0 (negative) indicates a benign file. Out of the total 14 malicious files, 11 were correctly classified as malicious (TP), while 3 files were misclassified as benign (FN). Out of the 14 total benign files, 13 were correctly classified as benign (TN), while one file was incorrectly classified as malicious (FP). An overall accuracy of 0.857% was achieved by the model.

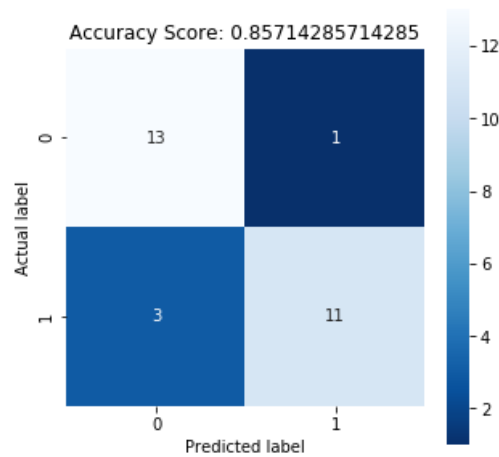


Figure 2.3 - Confusion Matrix with Accuracy

2.4.2 Accuracy, Specificity, Precision and Recall

Accuracy is the measure of predicted TPs over actual TPs. Accuracy by itself is not the best measure to use when gauging the precision of an ML model. For example, if testing to detect cancer in 1000 patients, the model's output predicted that no patient had cancer, but in fact 50 patients did, the model would have an accuracy of 95%. From the previous example, it illustrates how the accuracy metric does not by itself provide a clear indication of performance as accuracy is not the only measure which needs to be taken into consideration when evaluating the performance of a ML model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Specificity measures the proportion of predicted negatives which were true negatives. It evaluates the total amount of TN over the total TN and FP. Specificity is measured between 0.0 for no specificity and 1.0 for total specificity, with a higher value representing a greater percentage of predicted TN.

Using the values from Figure 2.3 as an example, the model produced 13 TNs and 1 FP, which equates to a specificity score of 0.928.

$$Specificity = \frac{TN}{TN + FP}$$

Precision is the number of TPs over the total TPs and FPs. It evaluates the total amount of TP over the total TP and FP. Recall, same as specificity, is measured between 0.0 and 1.0. Using the same values from Figure 2.3, the model produced 11 TPs and 1 FP, which equates to a precision score of 0.916.

$$Precision = \frac{TP}{TP + FP}$$

Recall is a method used in conjunction with precision to calculate the True Positive Rate (TPR) of a model. Recall is also known as sensitivity. The TPR represents the percentage of actual positive classifications. The TPR and False Positive Rate (FPR) will change depending on the threshold

$$Recall = \frac{TP}{TP + FN}$$

Precision, recall, and accuracy have been used to determine the quality of models by many researchers. As an example, Feroze, Baig, and Johnstone (2015) used precision, recall, and accuracy to compare the quality of two-tiered and single-tiered spam detection models. Accuracy was also used as the performance measure when comparing multilayer extreme learning machine and deep neural network models for intrusion detection applications (Yang, Wang, & Johnstone, 2020). The quality of the model is represented by the value produced from the chosen performance measure, which were described in the previous sub-sections. When comparing different models, the model with a higher value indicates greater performance than a lower value model.

2.4.3 Precision-Recall or PR Curve

A PR curve is a graph which plots the precision values on the y-axis and the recall values on the x-axis for different probability thresholds. The precision is related to the number of false positives and the recall is related to the number of false negatives, with a high precision and recall value representing a low number of false positives and false negatives. As illustrated in Figure 2.4, the precision and recall values are shown at different thresholds on the orange line, compared to a no skill model (50-50 chance of a TP) on the dotted blue line.

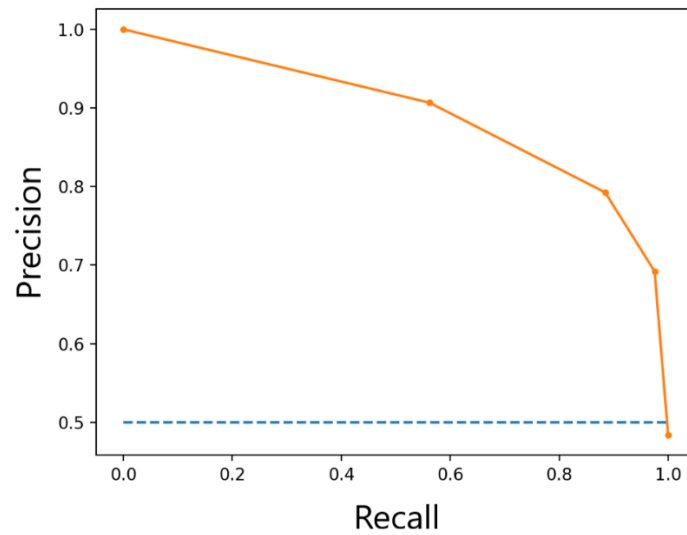


Figure 2.4 - PR Curve Example

A good ML model will aim to have both high precision and recall values, but it will usually sacrifice one or the other to some degree. The quality of the PR curve is represented by the Area Under the Curve (AUC) as shown in the multi-class example in Figure 2.5, with a higher value representing better model performance.

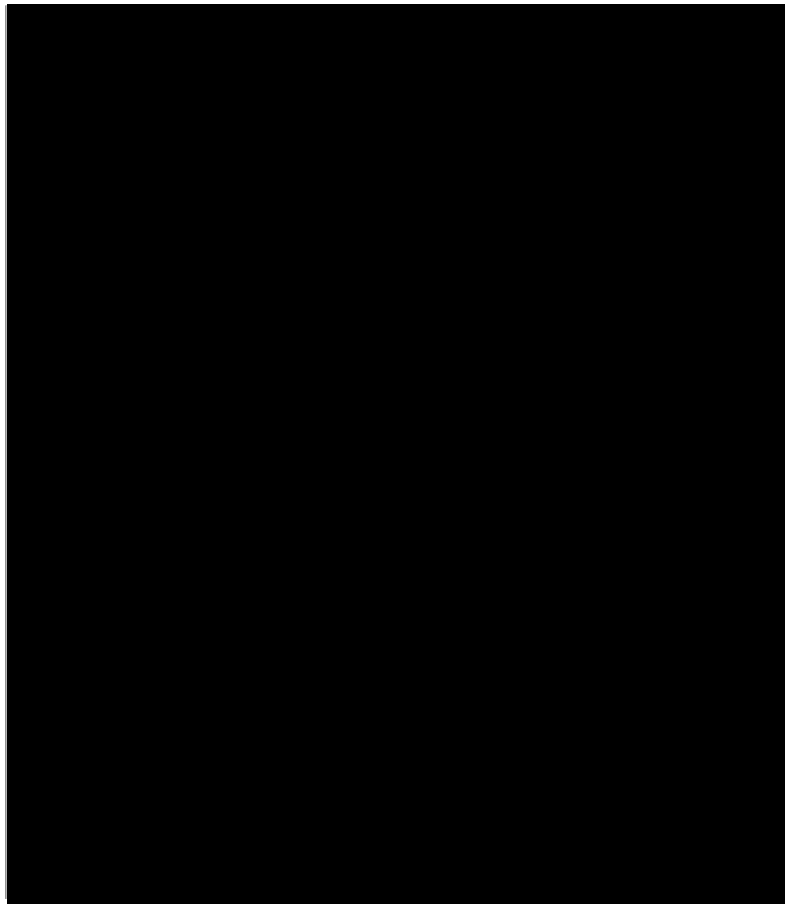


Figure 2.5 - Multi-class Precision-Recall Curve Example Adapted from Scikit
Refer to

https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

In Figure 2.5 the dark blue class is the best performer (as it has the greatest area under the curve), while the light blue class is the worst performer (as it has the least area under the curve). At perfect recall, each class has a similar precision value, with the light blue class having the greatest value. At zero recall, three of the classes have perfect precision, while the light blue class has zero precision. By examining the PR curves, a judgement can be made about what trade-off between precision and recall should be made to generate a model with the greatest TPR.

2.4.4 ROC (Receiver Operating Characteristics) Curve

The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR). The FPR is the ratio of incorrect positive classifications i.e., negative objects being incorrectly classified as positive. The ROC curve is used to summarise the performance of a binary classification model.

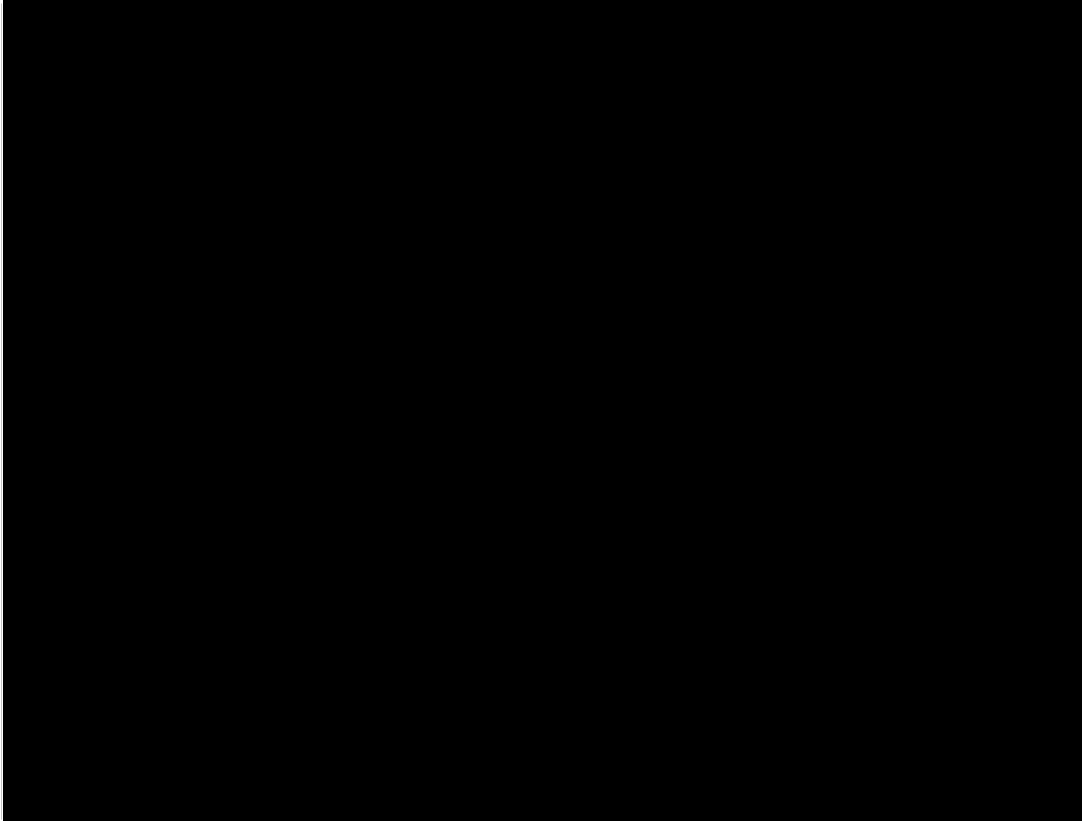


Figure 2.6 - ROC Curve Example

Refer to

ROC Curve Plot for a No Skill Classifier and a Logistic Regression Model

<https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/>

As it is illustrated in Figure 2.6, the dotted diagonal line represents a no skill model which produces a 50/50 chance of a true positive. The orange line representing a logistic model illustrates the performance of the model at different at different TPR/FPR values.

2.4.5 F1 score

The F1 score is the harmonic mean calculated from the precision and recall which is used to compare the performance of two or more classifier models. The harmonic mean treats low values with more importance, compared to the normal mean which treats all values equally. For a model to produce a high F1 score it needs to have a high level of both precision and recall.

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

The F1 score is a popular performance measure used in ML malware detection. The F1 score was used by Abhishek and Goswami (2017) in their Android malware detection research, by W.-C. Wu and Hung (2014) in their Android malware detection, and by F. Xiao, Lin, Sun, and Ma (2019) in their ML malware detection framework for IoT environments.

2.4.6 Matthews Correlation Coefficient

Matthew Correlation Coefficient (MCC) is a measure for determining the quality of binary classification models in ML. The MCC formula uses the TP, TN, FP, and FN values to calculate the efficiency of the model. An MCC can be calculated from a confusion matrix by using the following formula:

$$MCC = \frac{(TP * TN - FP * FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The MCC produces a value between -1 and +1. A value of +1 represents a perfect prediction, -1 represents a model which produces each output incorrectly, and a value of 0 indicates that the model is no better than a random prediction of the classification.

According to Chicco and Jurman (2020), MCC is a more accurate measure for binary classification ML models compared to the F1 score, even though the F1 score is currently a more popular choice. In the domain of IoT botnets, Peacock (2019) found that MCC was a better metric overall, compared to the previous metrics, when used for comparing models which were trained for detecting anomalous network traffic generated by IoT botnets.

2.5 Algorithm Overview

Several ML algorithms which have been utilised in the cyber security domain are described in the following subsections. As this research pertains to the effectiveness of ML applications in a specific sub-domain (malware detection) and not the development of ML algorithms, the following subsections provide an overview of the principles of each candidate ML algorithm.

2.5.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are ML systems loosely based on how biological neural networks (e.g., brains) learn via pattern recognition (Haykin, 2008). There are different types of ANN such as the Feedforward Neural Network (FNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Probabilistic Neural Network (PNN). ANNs can be used in either supervised, unsupervised, or reinforcement scenarios. The underlying structure in each of the different variants of ANNs is identical.

ANNs use a collection of nodes called artificial neurons which are information-processing units. The neurons transmit weighted signals between each other through connecting links called synapses. The weights of the signals are summed along with a bias value and are passed through an activation function. The activation function determines the activation state of the neuron, if the neuron is activated, it will transmit the data to neuron(s) in the next layer. In an ANN, a collection or set of neurons are organised together to comprise a layer. In general, a neural network will be comprised of an input layer, one or multiple hidden layers, and an output layer (Haykin, 2008).

ANNs are a popular ML algorithm. They have been used for various visual recognition tasks (Y. Guo et al., 2016), speech recognition (W. Liu et al., 2017), Google uses NNs in their language translation software (Y. Wu et al., 2016), and the development of autonomous vehicles (Tian, Pei, Jana, & Ray, 2018).

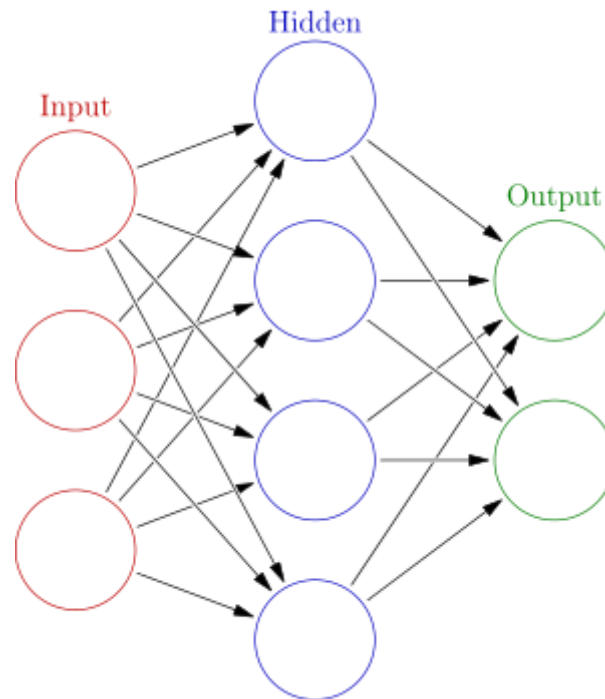
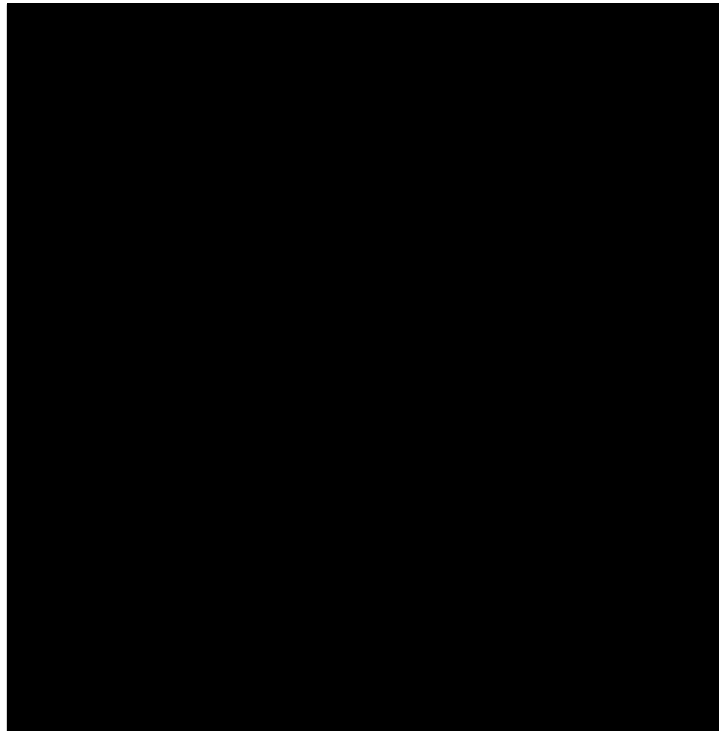


Figure 2.7 - Diagram of a Feedforward ANN Adapted from ("Artificial neural network with layer coloring," 2013)
Retrieved from
https://en.wikipedia.org/wiki/File:Colored_neural_network.svg
CC BY 3.0

2.5.2 Support Vector Machines

Support Vector Machines (SVMs) belong to the ANN category; they can be used in supervised and un-supervised learning approaches for both classification and regression problems. For classification problems, SVMs use a non-probabilistic linear model to classify data into binary classes. Multiple binary classification models can be used to solve multi-class classification problems (Bishop, 2006). SVMs can also be used to solve nonlinear regression problems using the kernel method (Rogers & Girolami, 2016).

For solving a classification problem, an SVM, after receiving a set of labelled training data, will generate an optimal hyperplane which classifies new input data into the correct binary class. The optimal hyperplane is found by determining the greatest margin achievable to split the data into the appropriate classes. The margin is calculated from the sum of the distance between the closest data point from each side of the hyperplane. The data points used to generate the hyperplane are known as the support vectors, as illustrated in Figure 2.8.



*Figure 2.8 - Optimal hyperplane of a linear SVM. Support vectors are coloured blue. Adapted from Haykin (2008)
Refer to Figure 6.1 Illustration of the idea of an optimal hyperplane for linearly separable patterns
Haykin, S. (2008). Neural Networks and Learning Machines (Third ed.). Upper Saddle River, New
Jersey: Pearson.*

2.5.3 Decision Trees

Decision trees are a supervised learning method which are used in ML as a predictive model for both classification and regression problems (Alpaydin, 2004). Decision trees are populated with nodes, which contain questions. The questions relate to a feature of the input data, the answer to which is connected to another node. Questions can be either binary or contain multiple variables. For multiple variables, a weight is calculated to determine which answer is most accurate. Input data fed into decision trees start at the root node, and progress through a path of nodes to determine how the data should be labelled. The node path is obtained by answering the questions.

Multiple decision trees can be used together to achieve a more accurate result compared to the use of one decision tree. Random Forest (RF) and Boosted Trees (BT) are two methods that use an ensemble of decision trees to obtain a more accurate result, compared to single trees (Banfield, Hall, Bowyer, & Kegelmeyer, 2007). In the RF approach, multiple decision trees are run in parallel to each other and the final result is determined from the average result of each decision tree in the ensemble. In the BT approach, a sequential stream of decision trees is trained using the results from the previous trees to populate the nodes. The final result obtained through BT is the weighted average from each decision tree in the ensemble.

2.5.4 Regression Models

Regression is a technique developed for use in statistics to identify the relationship between independent (predictor) and dependant (output) variables. Regression techniques can be used in a supervised learning approach to solve continuous problems (Linear Regression) or discrete problems (Logistic Regression) (Bishop, 2006). There are different types of linear regression techniques, such as Simple Linear Regression when using a single independent variable, or multivariate regression when using multiple independent variables.

Logistic Regression is a supervised classification approach which is used to solve binary classification problems. Multiple logistic regression models can be used together to classify variable data from a multiclass classification problem. To solve a multiclass problem, either a one-versus-the-rest (OvR) or a one-versus-one (OvO) approach is used (Bishop, 2006). In an OvR approach, one classifier is trained for each possible outcome in the multiclass problem. The classification is determined by testing the input against each classifier and choosing the one which has the greatest classification score. In an OvO approach, a classifier is trained for pairs of possible outcome classes. The OvO approach reduces the number of classifiers that need to be trained and therefore the computational power used.

2.5.5 Naïve Bayes

Naïve Bayes is a supervised probabilistic-classification ML algorithm. In a Naïve Bayes approach, the probability of each feature is considered independent of each other feature. That is, when determining which class a variable belongs to, the probability of a feature being true does not increase or decrease the probability of another feature. Different probability algorithms can be used in Naïve Bayes, with the Bayes Theorem being most common.

Naïve Bayes theorem calculates the probability of each factor independently, then selects the factor with the highest probability of occurring. Naïve Bayes assumes each factor is independent from each other.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Naïve Bayes is a popular algorithm for use in spam detection. It is used in SpamBayes, SpamAssassin, and Bogfilter (Nelson et al., 2008). In a Naïve Bayes spam filter, features of an email (e.g., headers and keywords) are assigned a probability value from the likelihood that the keyword word appears in a spam email. The probability is determined through training of an ML model in a supervised manner where spam and ham emails are pre-labelled, ham being email that is not spam, i.e., legitimate email. Each feature's probability score is determined independently from the other features. The probability of an email being spam is determined by analysing each feature and determining if the probability is greater than the spam threshold set by the application.

2.5.6 K-means Clustering

K-means clustering is an unsupervised clustering algorithm which aims to determine groupings of data from an unlabelled dataset (Chio & Freeman, 2018). The numbering of the groups is represented by the variable K , and each group K is centred on the mean of a cluster of data points. The K-means algorithm automatically determines the groupings through an iterative function (Chio & Freeman, 2018). First, a random selection of centroids are chosen for data points to aggregate around. Second, the mean of the data points surrounding the centroid from each K group is determined, this mean point is then selected as the new K th group's centroid point. The previous process is repeated until the centroid position of each K th group no longer changes, which produces the final output of the found clusters from the input data (Chio & Freeman, 2018).

K-means clustering is popular in the business domain for grouping together clusters of purchases made by consumers, which benefits businesses in determining which items to sell together or what type of deals are more likely to be popular. In the cyber security domain, K-means clustering has been used in general malware detection (Martin, Menéndez, & Camacho, 2016), Android malware detection (D. J. Wu, Mao, Wei, Lee, & Wu, 2012), botnet detection (Dietrich et al., 2011), and network intrusion detection (Jianliang, Haikun, & Ling, 2009).

2.6 Machine Learning Applications

Machine learning has been utilised in a variety of domains to automate processes and perform complex data analysis, which would take a significantly longer time if being performed manually (Obulesu, Mahendra, & ThirlokReddy, 2018). The following sub-sections provide a brief overview of the types of domains which have utilised ML with some practical examples of ML in use.

2.6.1 Critical Infrastructure

Machine learning can be utilised to provide security for a nation's critical infrastructure. The securing of critical infrastructure is paramount for the wellbeing and day-to-day operation of any nation. In 2015, a malware attack targeted the SCADA systems of three utility companies in the Ukraine, which left thousands of Ukrainian homes without power for several hours (Allianz, 2020). The utilisation of ML to secure critical infrastructure and to maintain operation of the system can be utilised at both the network and human level of operation (Zeadally, Adi, Baig, & Khan, 2020). The human element is susceptible to phishing attacks which can allow for malware to be executed on critical infrastructure systems. ML trained phishing detection applications can be utilised to prevent the phishing attacks from succeeding. ML can be utilised at the network level to identify anomalies within the operation of the critical infrastructure system, such as high load operation at odd times or low operation at peak times.

2.6.2 Computer Vision

Computer vision is a field of research in computer science which aims to provide computers with a method of understanding digital images and videos in the same way that a human can. Computer vision is used in a variety of applications in varying fields such as military, medical, and industrial manufacturing. The following is a general list of applications which utilise computer vision in some way:

- Facial Recognition
- Species Identification
- Product Quality Examination
- Autonomous Vehicles
- Medical Image Processing
- Missile Guidance

The method of computer vision is dependent on the type of application it is being utilised in but some functions, such as image acquisition, data pre-processing, feature extraction, high-level processing and decision making are generally used by the majority of computer vision applications (Janai, Güney, Behl, & Geiger, 2020). In the case of an autonomous vehicle, computer vision may be utilised in the following way. The image acquisition is obtained by a set of cameras attached to the vehicle which are recording digital images/video. The acquired digital images are pre-processed to fit into certain dimensions or to remove excess noise from the image. The feature in the image is identified by using a trained ML model to classify the objects. A decision is made based on the objects identified in the image e.g., the vehicle stops as it has identified a stop sign at the next intersection.

2.6.3 Natural Language Processing

Natural language processing (NLP), also known as computational linguistics, is a sub-field of computer science, statistics and linguistics. Natural language processing is used to describe how the problem of understanding the structure of human language in a computer-oriented view is solved (Belinkov & Glass, 2019). NLP is used in many applications such as predictive text, automated translation, speech recognition and sentiment analysis (Young, Hazarika, Poria, & Cambria, 2018). Predictive text is used to predict the next word which could be used to complete a sentence being typed based on the context of the previously typed words (Young et al., 2018).

2.6.4 Cyber Security

In the cyber security domain, ML is utilised in a multitude of various applications. Cyber security measures benefit greatly from the precision and speed of automation provided by machine learning-based applications. The following is a general list of applications in cyber security which utilise machine learning:

- Malware Detection
- Network Intrusion Prevention
- Network Intrusion Detection
- Biometric Authentication
- Spam Detection

2.6.5 Finance

In the financial field, machine learning is utilised in the following areas:

- Stock Prediction
- Fraud Detection
- Personal Finance Management (Budgeting)
- Targeted Personalised Advertising
- Financial Risk Analysis
- Credit Analysis

Machine learning is used in finance to analyse large amounts of financial data and produce risk analyses on portfolios, and potential earnings (Shen, Jiang, & Zhang, 2012). Machine learning is also utilised for anomaly detection, which automates the process of detecting fraud (Perols, 2011). Machine learning algorithms can build models on a user's traditional purchasing patterns; alerting the user when an anomalous pattern is detected (Raj & Portia, 2011). Fraud detection is utilised by banking institutions to identify anomalous transactions within a customer's purchase history which may indicate that some type of fraudulent process had occurred. Fraud detection utilises anomaly detection from ML to identify transactions which lie outside of the predicted purchasing pattern a customer would generally follow.

2.6.6 Marketing

Marketing utilises ML to identify patterns within people's behaviour which can be used to generate advertisements, both general and targeted, and to optimise product placement to increase sales. Personal advertisements created from online behaviour are tailored to a particular user to provide them with an option to purchase an item/service which the advertisement company believes the user is interested in based upon their browsing habits.

2.6.7 Healthcare

In the medical field, machine learning is utilised in the following areas:

- Medical Diagnosis
- Precision Medicine
- Medical Data Collection

- Autonomous Surgery
- Drug Development/Discovery
- Disease Identification

IBM Watson Genomics, a partnership between IBM and Quest Diagnostics, aim to improve precision medicine (PM) by utilising machine learning algorithms. PM is a tailored approach for providing healthcare to individuals, mainly disease treatment and prevention, which is achieved by utilising the information known about the individual's genetic makeup, lifestyle, and environment. Machine learning is also used for the general population, for instance, ML is used to analyse large datasets of different publicly available data to predict the likelihood that an epidemic outbreak may occur.

2.6.8 Personal Assistants

Siri, Cortana, Alexa, Echo and Google Assistant are Smart Personal Assistant (SPA) applications which utilise machine learning. SPAs perform tasks such as organising dates and reminders, scheduling appointments (Google, 2018), performing Internet searches, activate music and video players, creating to-do lists and even purchase items online. Netflix and other streaming services utilise ML to identify which type of show a user may be interested in viewing based on their browsing history and the history of other users with similar viewing habits.

2.7 Adversarial Machine Learning

Machine learning algorithms are susceptible to a range of adversarial attacks. Adversarial attacks aim to, in some manner, reduce the efficacy of an ML based application. This is not a new idea, over a decade ago, Barreno, Nelson, Sears, Joseph, and Tygar (2006) proposed a taxonomy of adversarial ML which classifies adversarial attacks through a spectrum of three axes: influence, specificity, and security violation. The taxonomy was expanded by Huang, Joseph, Nelson, Rubinstein, and Tygar (2011), to include a framework for quantitatively evaluating security threats, which are shown in Table 2.

The influence axis is divided into two categories: Causative and Exploratory (Huang et al., 2011). A causative attack is when the attacker has a degree of influence over the training data, which is leveraged to alter the training process to benefit the attacker (Huang et al., 2011). This type of attack is also called a poisoning attack. An exploratory attack is when the attacker is not able to influence the training process, and instead through an exploratory examination of the model (e.g., offline analysis) aims to gather information on how the model operates, e.g., feature detection (Huang et al., 2011).

The specificity axis is divided into two categories, Targeted and Indiscriminate (Huang et al., 2011). In a targeted attack model, the end goal of the attacker is focused on a known set of target points (Huang et al., 2011). In an indiscriminate attack, the end goal is a general attack which reduces the performance of the model overall (Huang et al., 2011).

The security violation axis is divided into three categories, Integrity, Availability and Privacy (Huang et al., 2011). An integrity-based attack aims to sabotage the integrity of the model, increasing the rate of false-negative results (Huang et al., 2011). An availability attack expands upon the integrity attack, where the attacker increases the rate of false negatives and false positives to such a degree, that the model is rendered completely unusable (Huang et al., 2011). In a privacy-based attack, the attacker wishes to gain information about a system's users (Huang et al., 2011).

Table 2 - Adversarial Attack Spectrum. Adapted from Huang et al. (2011)

Influence	Specificity	Security Violation		
		<i>Integrity</i>	<i>Availability</i>	<i>Privacy</i>
Causative	Targeted	Allow a specific intrusion	Render specific aspect of application unusable	Reveal targeted information
	Indiscriminate	Allow an intrusion	Render application unusable	Reveal information
Exploratory	Targeted	Discover an intrusion vector from a list of possible paths	Locate a set of misclassification points	
	Indiscriminate	Discover an intrusion vector		

The different levels of understanding an attacker can have about a target system, influence the type of attacks which can be performed. The different attacks which can be performed are categorised under three attack scenarios, white-box, grey-box, and black-box (Battista Biggio & Roli, 2018).

In a white-box attack scenario, the attacker has complete knowledge of the target system, this includes the training model, the feature set and the training data (Battista Biggio & Roli, 2018). White-box attacks represent worst-case scenarios and are unlikely to be performed in the real world. In a grey-box attack scenario, the attacker has limited knowledge of the target system model (Battista Biggio & Roli, 2018) For example, an attacker may know that a target system uses a decision tree algorithm for

training but has either limited or no knowledge of the labels being used. An attacker can utilise his/her limited knowledge to build a substitution model which more accurately represents the target system, compared to a black-box attack. In a black-box attack scenario, the attacker has zero knowledge of the target system model before developing an attack (Battista Biggio & Roli, 2018).

Frameworks for the development of attacks based upon the taxonomy of Huang et al. (2011) and the level of knowledge an attacker has have been explored by H. Xiao et al. (2015) and Suciu, Marginean, Kaya, Daume, and Dumitras (2018). Suciu et al. (2018) introduced the FAIL framework which identifies four classes of knowledge an attacker may have, which are *Feature knowledge*, *Algorithm knowledge*, *Instance knowledge* and *Leverage*. Feature knowledge refers to the features known to the attacker. Algorithm refers to the algorithm utilised by the attacker to craft adversarial examples. Instance refers to the labelled training data known to the attacker. Leverage refers to the subset of features which can be modified by the attacker.

A survey of different adversarial ML attacks and defences was undertaken by Liu et al. (2018), which categorised the papers according to the taxonomy developed by Huang et al. (2011). The attacks were categorised as poisoning, evasion, impersonation, and inversion attacks.

A poisoning attack is a causative attack performed at the training stage, which aims to reduce the classification accuracy of a target system by infecting the training dataset with malicious data. The end result of a poisoning attack may be to allow for certain data to bypass identification at test time, or to perform a denial of service attack, rendering the application inoperable (Huang et al., 2011).

Poisoning attacks are only feasible if an attacker is able to infect the training dataset with adversarial examples. As malware detection applications which utilise ML need to be constantly retrained with new data to keep up with the newly emerging malware variants (Gibert, Mateu, & Planes, 2020), this provides an attack vector for poisoning attacks. Malware authors can detect features of anti-malware software by modifying certain parameters of malware files, feeding the files to the anti-malware software and analysing the results to determine which aspects of the modified malware file were detected (Hu & Tan, 2017).

Evasion attacks aim to avoid detection by bypassing the feature detection of an ML system using maliciously crafted data samples, known as adversarial examples. Evasion attacks have been developed to bypass image recognition (Papernot et al., 2017), biometric authentication (B. Biggio, Fumera, Russu, Didaci, & Roli, 2015) and malware detection (L. Chen, Ye, & Bourlai, 2017).

Impersonation attacks aim to craft adversarial examples which can be used to fool an ML classifier. Impersonation attacks allow an attacker to bypass security mechanisms such as facial recognition systems (Sharif, Bhagavatula, Bauer, & Reiter, 2016). Inversion attacks are an exploratory privacy

attack, which aim to uncover private data, be it from the training data (e.g. medical data) or information about the training algorithm used (Fredrikson, Jha, & Ristenpart, 2015).

The key findings discussed by Q. Liu et al. (2018) are:

- New threats are constantly emerging and evolving
- Defensive adversarial security measures are at the initial stage of development
- Data privacy is key for protecting ML algorithms
- Secure deep learning is of key interest for future research
- Balance is required to achieve effective secure ML with little overhead

The following sub-sections cover different adversarial attacks which have been developed where the target system was an ML-based malware detection application or a spam detector. Other adversarial attacks which target ML systems that are not the focus of the research are not discussed.

2.7.1 Adversarial Attacks

The following sub-sections cover a variety of adversarial ML attacks for both poisoning and evasion, with a focus on attacks which target ML trained malware detection applications. The attacks discussed in this section cover a variety of ML algorithms and illustrate the different methods possible for generating adversarial examples. Following the attack section, an overview of different defensive strategies which have been developed to combat the variety of adversarial ML attacks is presented.

2.7.1.1 Adversarial Poisoning Attack (Spam)

The earliest research on poisoning attacks was produced by Wittel and Wu (2004). Their research covered performing a poisoning attack to reduce the efficacy of statistical spam filters. Since then, an assortment of different adversarial attacks have been developed for different ML-based applications covering a variety of ML algorithms.

Nelson et al. (2008) developed two adversarial poisoning attacks against the ML-based spam detector, SpamBayes. The two attacks developed were a dictionary attack and a focused attack. The goal of the dictionary attack was to perform an indiscriminate Denial-of-Service attack, rendering the spam detector unable to function at all. The goal of the focused attack was to perform a targeted integrity attack, in which a certain spam email could bypass detection while the general detection capabilities are left unaffected.

Both attacks were performed under the assumption that an attacker is able to introduce his/her adversarial spam emails into the training dataset, which is used to retrain the classifier to be able to detect new spam variants every week (Nelson et al., 2008). The attack was restricted in that email headers of the attack emails were not to be modified and each attack email was to be trained as spam (Nelson et al., 2008). The indiscriminate dictionary attack achieved a 36% misclassification rate, with

1% of the training data being adversarial examples. The targeted attack altered the classification of 60% of the targeted emails (Nelson et al., 2008).

2.7.1.2 Adversarial Poisoning Attack (Malware)

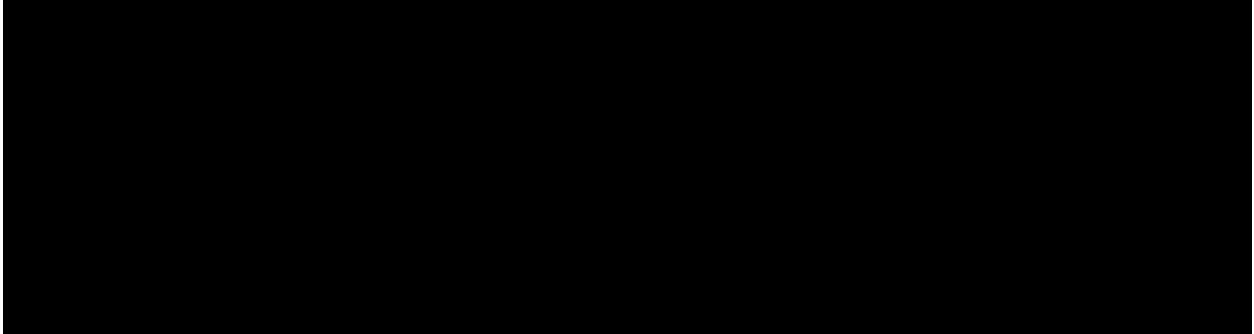
The first poisoning attack for malware detectors was performed by Newsome, Karp, and Song (2006). In their research, they proposed two types of poisoning attacks, the red herring attack and the inseparability attack. The red herring attack involves adding fake features to malware files, which will be removed by an attacker when performing an attack. The malware files containing fake features are fed to the ML algorithm and are intended to cause the learner to build a classification model which will identify files with fake features as malicious. When the attacker removes the fake features, the new malware file should bypass the malware detection as the file does not now contain enough features (or markers) to identify it as malware. The inseparability attack is a Denial-of-Service attack, where the attacker incorporates features from the benign training data into the malicious data set, which renders the classifier incapable of deducing at an acceptable accuracy, which files are malicious or benign.

StingRay is a framework developed to generate adversarial examples for targeted poisoning attacks, which was developed by Suciu et al. (2018). The aim of a StingRay attack is to induce either a false positive or false negative classification of a targeted example, while retaining the general classification of the ML model so as to not arouse suspicion that an attack has taken place. The attack scenario assumes that periodic retraining of data is undertaken to update the ML classifier (e.g., spam and malware detection), which is how the attacker introduces adversarial examples into the target system.

The StingRay attack was evaluated from attacking four different classifiers: image classification, Android malware detection, Twitter-based exploit prediction and data breach prediction. The ML classifiers were trained using Convolutional Neural Networks (CNNs), linear Support Vector Machines (SVMs) and Random Forest (RF). The Android malware detection application was trained using a linear SVM and the Drebin Android dataset, which is comprised of 123,453 Android applications, of which 5,560 are malicious. The Android attack performed was developed to induce a false negative classifying a targeted malicious Android application as benign. For the attack, a targeted malicious application which was correctly classified was selected and the identified malicious activity was recorded (e.g., API calls, suspicious URL requests and unauthorised permissions), the malicious features were introduced into benign applications which were fed to the learning algorithm. The performance of the Stingray attack was calculated from the success rate and Performance Drop Ratio (PDR) of each attack. The PDR is calculated by evaluating either the average accuracy or F1 score on a separate testing set. The StingRay attack on the malware classifier had a 50% success rate and a performance drop ratio of 0.99. The results from the StingRay attacks and

applied defensive strategies on the different ML applications is shown in Table 3, the defensive strategies are discussed in 2.7.2.

Table 3 - StingRay Attack Results (Average Instances, Success Rate, and Performance Drop Rate) Adapted From (Suciu et al., 2018) Refer to Table 6 Effectiveness of StingRay and of existing defenses against it on all publications
<https://arxiv.org/abs/1803.06975>



2.7.1.3 DNN Evasion Attack

Grosse, Papernot, Manoharan, Backes, and McDaniel (2016) developed an adversarial evasion attack against malware detection applications trained on feed-forward neural networks to detect static features. The attack was based on the method used by Papernot et al. (2017), which generated adversarial examples to bypass ANN trained image classification. The malware detection models generated were trained using a variety of parameters including, number of neurons (10, 50, 100, 200, and 300), number of layers (1- 4) and malware/clean application ratio (0.1, 0.2, 0.3, 0.4 and 0.5) combinations using the DREBIN Android malware data set. The baseline used to compare to other NN malware detectors were comprised of 2 layers of 200 neurons each. The baseline model achieved a 97% accuracy, with 7% false negatives and 3.3% false positives. The adversarial attack performed had varying degrees of misclassification depending on the size of the network and the malware ratio used. The highest misclassification rate achieved was 81.89% on a model comprised of one 200 neuron layer trained with a malware ratio of 0.4, and the lowest misclassification rate was 60.02% on a model comprised of two 50 neuron layers and a malware ratio of 0.4.

2.7.1.4 Malware Generative Adversarial Network (MalGAN) Attack

MalGAN is a Generative Adversarial Network (GAN) attack developed by Hu and Tan (2017), which generates adversarial examples for the purpose of bypassing ML-based malware detection models. A GAN is comprised of a generative model and a discriminative model, which work together to generate the optimal adversarial examples, an example is shown in Figure 2.9. The generative model creates the adversarial examples and the discriminative model determines which adversarial examples would be successful in an attack (Goodfellow et al., 2014).

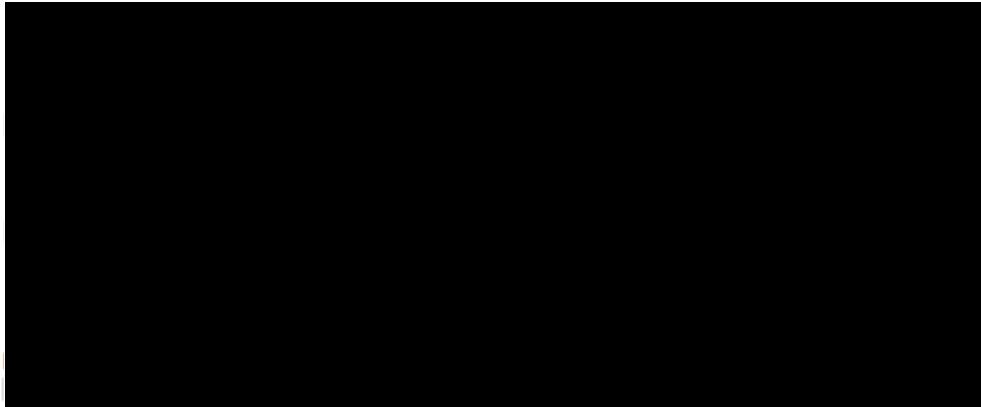


Figure 2.9 - The architecture of MalGAN adapted from Hu and Tan (2017)

Please refer to

Hu, W., & Tan, Y. (2017). Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. ArXiv e-prints. Retrieved from <https://arxiv.org/abs/1702.05983>

The discrimination model in MalGAN is a substitution detector developed to imitate the target ML malware detection application. The target ML malware detection application is a black-box system. The attackers do not have access to the training dataset used or have any knowledge of the target system's structure. The only information obtained by the attackers is if a file is identified as benign or malicious. The results from the target black-box are used to label the dataset on which the substitution detector is trained. The MalGAN generative model generates the new adversarial examples which are fed into the substitution model to determine what features changes are required to bypass detection.

The ML algorithms tested by Hu and Tan (2017) were random forest (RF), logistic regression (LR), decision trees (DT), support vector machines (SVM), multi-layer perceptron (MLP), and a voting-based ensemble of each of these classifiers (VOTE). Hu and Tan (2017) claim that MalGAN can reduce the detection rate to near zero for each model and will require retraining with knowledge of the adversarial MalGAN examples.

The results from the MalGAN attacks on the different ML algorithms are shown in **Error! Reference source not found.** and Table 5. **Error! Reference source not found.** contains the results from the transferability version of the MalGAN attack, which was performed using adversarial examples which were generated using training data from a different dataset. Table 5 contains the results from the MalGAN attack which generated the adversarial examples from the same training dataset for each ML model.

Table 4 - MalGAN Adversarial Attack Results (Different Training Set) Adapted From -Hu and Tan (2017)

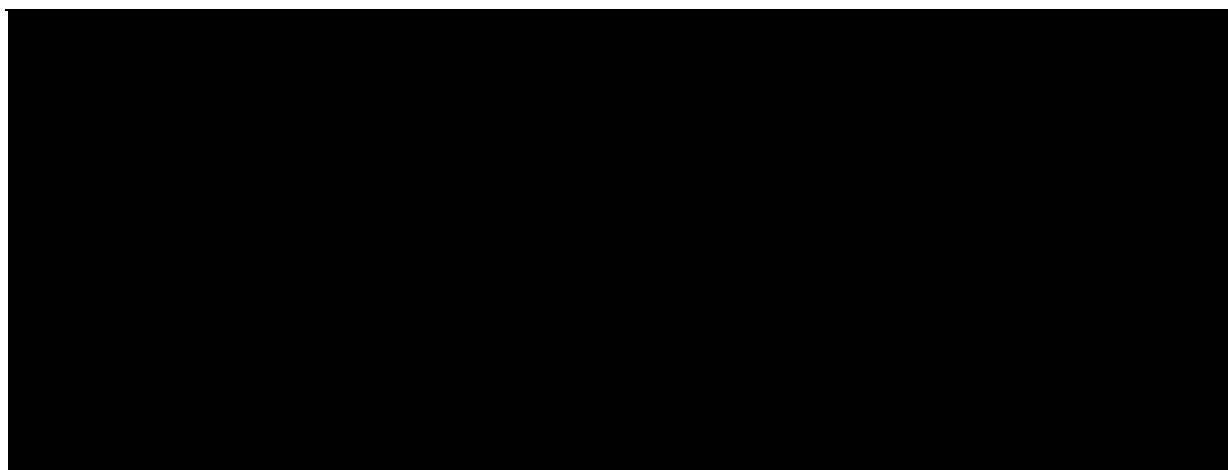
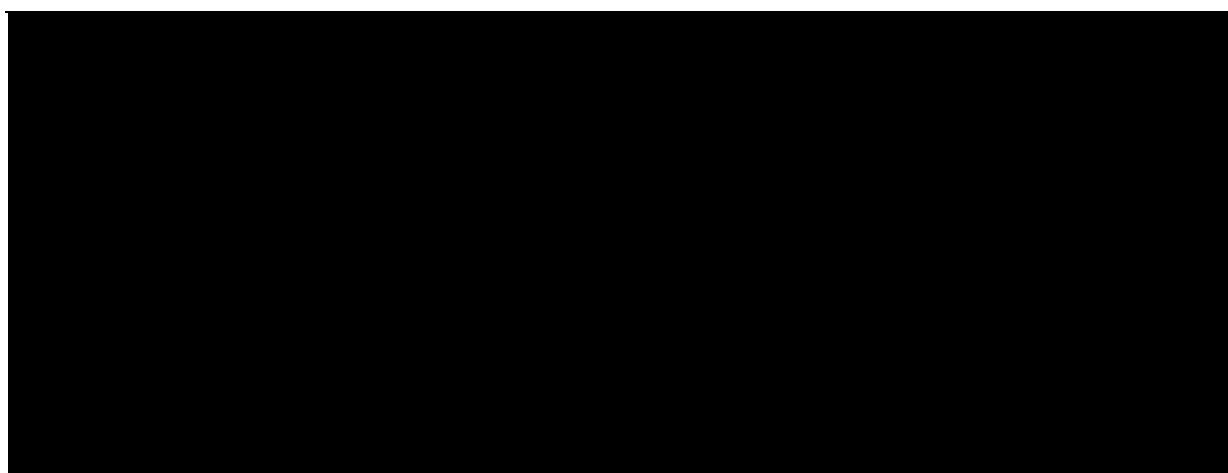


Table 5 - MalGAN Adversarial Attack Results (Same Training Set) Adapted From -Hu and Tan (2017)



For Table 4 & 5 Refer to

Hu, W., & Tan, Y. (2017). Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. ArXiv e-prints. Retrieved from <https://arxiv.org/abs/1702.05983>

2.7.1.5 Microsoft PE Evasion Attack

Anderson, Kharkar, Filar, and Roth (2017) produced a black-box based attack to bypass static malware detection of malicious Windows PE files. The attack was developed under three assumptions:

- No knowledge of the ML classifier architecture (features, structure, parameters)
- Can only receive a binary result from the classifier (malicious or benign)
- No use of an oracle to determine if generated PE retains original functionality

The assumptions were chosen to perform an attack in which the authors believed to be the most difficult scenario an attacker may face.

A Deep Reinforcement learning approach was chosen as the basis of the attack. The attack performed is based on a game scenario, where in each turn, a random modification is made, and a result is collected from the classifier. If the classifier identifies the file as malicious, another round is played,

up to a maximum 20, otherwise a successful attack is recorded if the file was identified as benign before the round limit.

The modifications made to the malicious Windows PE file do not alter the original code execution or break the PE file format (e.g., alter a name section, append bytes to end of sections and pack or unpack the PE). The black-box attack achieved a successful evasion rate of 16%.

2.7.1.6 Adversarial API Call Evasion Attack

A black-box attack was developed by Rosenberg, Shabtai, Rokach, and Elovici (2018) to evade detection from ML-based malware classifiers which have been trained to detect malware via analysing API call sequences. The authors proposed a framework, GADGET (*Generative API aDversarial Generic Example by Transferability*), which generates adversarial malware by adding new API calls to malware files. No removal or modification of original API calls is made to ensure original functionality of the malware is not altered.

The attack uses a white-box substitution model to craft effective adversarial examples which are transferred to attack black-box models, as it has been shown that adversarial examples succeed when transferred to attack ML models which have been trained on different datasets (Suciu et al., 2018; Szegedy et al., 2014). The substitution model is developed as the attacker does not have any knowledge of the ML algorithm or dataset used to train the target black-box malware detector. The only information it is assumed the attacker knows, is the API calls used in the target malware which will be modified into adversarial examples.

The Rosenberg, Shabtai, Rokach, et al. (2018) GADGET attack was performed against a variety of malware detection models trained using different ML algorithms. The attack was shown to be successful, with a 100% success rate against RNN and a majority of the attacks achieving over 90% success. The least successful attack was against Logistic Regression at 69.73%.

Rosenberg, Shabtai, Elovici, and Rokach (2018) developed another black-box attack framework, BADGER (*Benign API aDversarial Generic Example by Random perturbation*), which does not require the training of a substitute model, as the previous attack required, or any knowledge of the target ML malware detection model. The only assumption is that the attacker has knowledge of a subset of the API call sequence in the target malware and the attacker either knows the confidence score of the malware classifier or only the output label of the malware classifier. When the attacker knows the confidence score, the success rate of the attacks was around 98% and when the attacker only had the output label, the success rate of the attacks was 64%.

2.7.1.7 Subpopulation Data Poisoning Attacks

A form of poisoning attack which targets a sub-population inside of a large multi-class machine learning model was developed by Jagielski, Severi, Harger, and Oprea (2020). The attack operates in a black-box scenario, with the attacker not having knowledge of the exact model architecture or

parameters. The attack only targets one of the model's classes, the general efficacy of the other classes is unaffected by the attack. The attack was tested against the UCI Adult dataset, CIFAR-10 for image classification, UTKFace for face recognition, and IMDB reviews for sentiment analysis. The attacker does not have access to the original training data but does have a substitute training set which is generated from samples from the original set. The poisoning attack uses a label flipping approach with the target subpopulation being chosen from using "FeatureMatch" and "ClusterMatch" selection techniques. The FeatureMatch selection technique uses "fine-grained manual annotation" to identify subpopulations within the dataset (e.g., race or age in images of people). The ClusterMatch selection technique automates the manual annotation with clustering. By identifying clusters of subpopulations within the dataset, the attacker can select which subpopulation they wish to attack.

2.7.1.8 Watermark Attack (Malware)

Severi, Meyer, Coull, and Oprea (2020) developed a targeted poisoning attack by injecting a watermark into the benign training data, which influences the classification of the model to produce false negatives of malicious files if at test time they contain the same watermark.

The watermark attacks were performed on both GBDT and ANN models which used the EMBER dataset, which is a dataset of 800,000 Windows PE32 files. The watermark process does not impede on the original functionality of the adversarial binary file. The attack only required 8 features, out of 2351, to be modified for a successful attack. After injecting 1% poisoned (adversarial) samples into the training data, Severi et al. achieved a success rate of over 97%.

The watermark attacks were performed by identifying how the features contributed to the files classification by calculating their SHapley Additive exPlanations (SHAP) values. SHAP values are generated through a game theory approach where the feature vector values are replaced with other values in the feature vector set to identify their contribution i.e., if value A is replaced with value B and the classification shifts towards C or D, then value A had a positive contribution towards whichever classification. This process is computed for every feature vector. As EMBER uses feature hashing to compute the majority of its feature vectors (2,316 of 2,351), the watermark attack was performed on feature vectors which were not generated from the feature hashing process. The features modified for the watermark attack were evaluated on a set of malware files and benign software to ensure that the functionality of the files was not compromised. It is not possible to check that the modification would not break the files used to generate the EMBER dataset, as EMBER only contains the extracted features, not the complete binary.

Both white-box and black-box watermark attacks were performed. In the white-box scenario, the attacker had access to the original training data and model, which allowed for the attacker to generate accurate SHAP values. In the black-box scenario, the attacker only knew of the feature space the model was trained on but did not have access to the model's architecture. A transfer attack was performed,

where the attacker trained their own model, using the same feature space, and calculated the SHAP values of malware and benign software from that model. The calculated values were then used to identify which features should be manipulated in the benign training data of the target model.

Value selection is performed in two ways, MinPopulation and CountSHAP. In MinPopulation, values are selected from those which appear least frequently in the dataset. CountSHAP are the opposite to MinPopulation and are comprised of SHAP values which have a high density within the benign software.

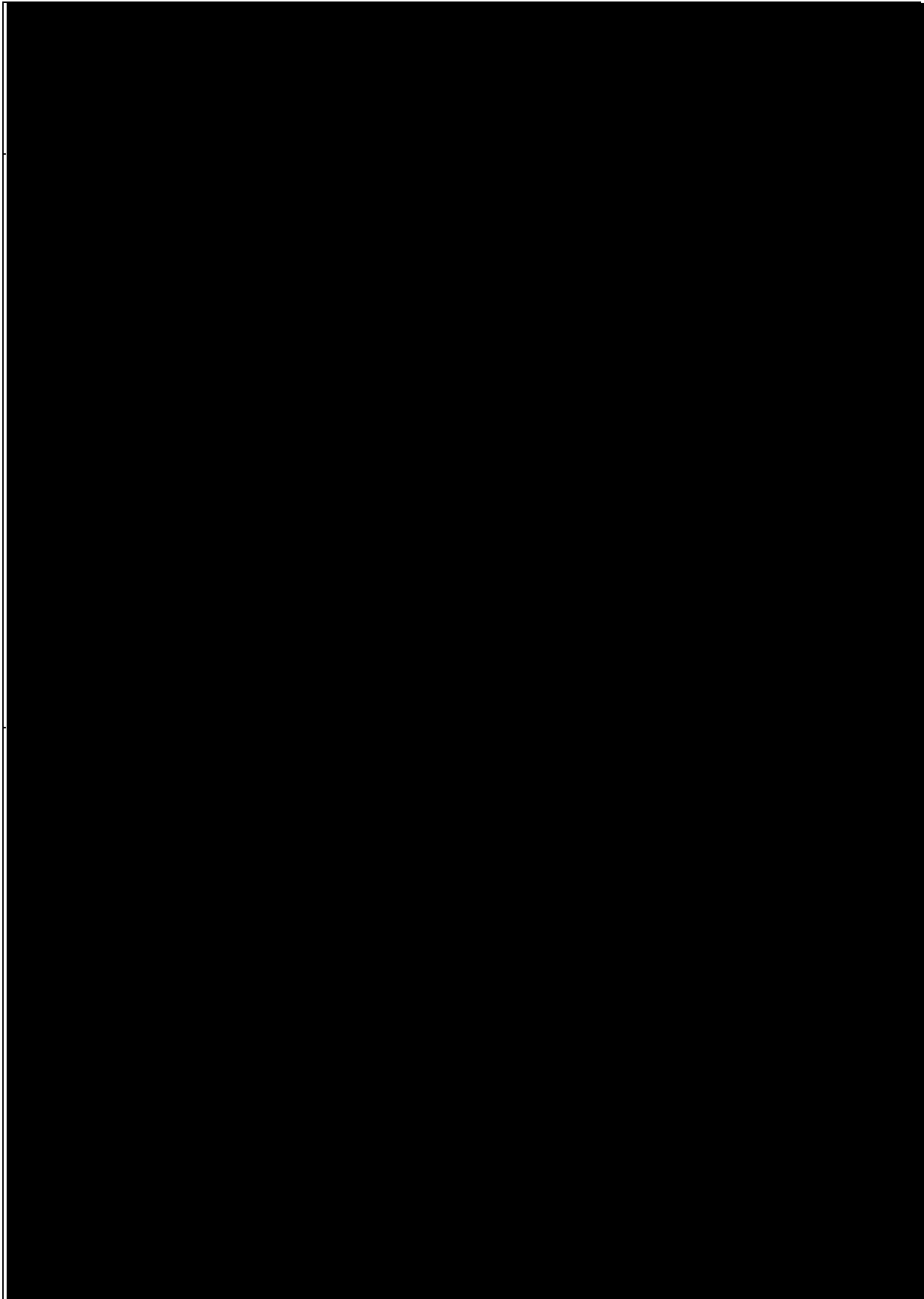
Three mitigation measures (Spectral Signatures, HDBSCAN, and Isolation Forest) were evaluated over the different models and attacks. The mitigation measures work under three assumptions, that the defender has:

- access to the poisoned training data
- has access to a clean labelled dataset
- believes that the attacker will attack the most relevant features of the model

All three mitigation measures are run on the training data to prevent an adversarial model from going live.

The defensive strategies were evaluated using a reduced feature space obtained from the 32 most important features from a clean portion (120,000 samples) of the training set. The values of the reduced subspace were all normalised to $[-1,1]$. The Spectral Signature defence worked by calculating the singular value decomposition of the benign samples in the new reduced feature space, then by calculating the outlier score and finally filtering out the top 15%. The Hierarchical Density-Based clustering (HDBSCAN) defence is influenced by the activation clustering defence by B. Chen et al. (2018) mentioned previously. The HDBSCAN defence works on the assumption that the watermarked samples generate a tight cluster of high density within the reduced feature space. The Isolation Forest is an unsupervised anomaly detection algorithm for identifying the watermarked samples as outliers with the reduced feature space. The results from the attack and defences are shown in Table 6.

Table 6 - Watermark Attack and Defence Results Adapted From (Severi et al., 2020)
Refer to: Table 3 in Severi, G., Meyer, J., Coull, S., & Oprea, A. (2020). Exploring Backdoor Poisoning Attacks Against Malware Classifiers. ArXiv e-prints. Retrieved from <https://arxiv.org/abs/2003.01031>



2.7.2 Adversarial Defence Strategies

Having described the various adversarial attack strategies which were found to be successful in influencing ML models, it is now appropriate to examine the defensive strategies which have been developed to combat the growing threat of adversarial ML attacks. The defensive strategies discussed cover a variety of ML algorithms and applications and do not solely concentrate on ML malware detection applications.

The defensive strategies developed to protect ML algorithms from adversarial attacks can be split into two categories determined by the underlying principle of the defensive strategy, which are the identification and removal of adversarial examples at the training stage (e.g., data sanitisation), and the development of robust algorithms which are resilient to adversarial attacks (e.g., adversarial training) (Battista Biggio & Roli, 2018).

2.7.2.1 Data Sanitisation

Data sanitisation is a training stage protection method which aims to detect and remove adversarial examples from a dataset to avoid poisoning the training algorithm. Data sanitisation is used in applications which rely on an open training model, where the learning algorithm needs to be periodically retrained to detect new emerging threats, such as spam detection and malware detection.

Reject on Negative Impact (RONI) is a data sanitisation defence developed by Nelson et al. (2008). RONI was developed to defend ML-based spam detectors from poisoning attacks at the training stage. The RONI defence evaluates the impact of new training data and discards the data if the impact is considered significantly negative by reducing the classification rate by 0.05 or greater.

The RONI defence was trained on five sets of training and validation data. The training sets were comprised of twenty independently selected emails from the Enron data set which was originally used to train SpamBayes for the spam attack in Section 0. The validation set was comprised of 50 emails and the query data was comprised of 120 random non-attack spam messages and 15 repetitions of seven variants of the dictionary attack. The training sets were trained with and without each of the query data samples. If the average impact across each validation set was severely negative then the query data was discarded, otherwise it was used in the training set for the next round.

The threshold of significant negative impact was determined from analysing the results of the adversarial dictionary attack, and the non-attack spam. The dictionary attack had an average negative impact of 6.8 ham messages being misclassified as spam, while the non-attack spam caused at most a 4.8 misclassification rate. Results from the experiment had RONI to be 100% effective against dictionary attacks, identifying and discarding each attack email and retaining each non-attack email for training (Nelson et al., 2008), however, RONI did not as well for focused attacks where the intention is to have a future targeted spam email bypass detection.

2.7.2.2 Robust Algorithm Approach

A variety of different defensive strategies have been proposed with the aim of developing robust algorithms which are resilient to the effects of adversarial attacks. Some of the methods proposed for developing robust algorithms include Adversarial Training (Goodfellow, Shlens, & Szegedy, 2015), the Ensemble Method (Tramèr et al., 2018), and Defensive Distillation (Papernot, McDaniel, Wu, Jha, & Swami, 2016).

Adversarial training is a defensive strategy proposed by Goodfellow et al. (2015) to increase the resilience of ML models from adversarial attacks. The general principle behind adversarial training is that including adversarial examples in the dataset will learn to detect adversarial examples, increasing the resilience of the generated ML model.

Ensemble adversarial training was proposed by Tramèr et al. (2018) to increase the resilience of an ML algorithm in comparison to the original adversarial training which was found to be vulnerable to transfer attacks. In Ensemble adversarial training, adversarial examples which have been trained to attack an ensemble of different models are introduced into the training dataset to increase resilience from adversarial attacks. As the transferability property has shown that adversarial examples trained on one model can be successfully transferred to attack another model even if the dataset used for training was different (Szegedy et al., 2014), the extended adversarial training defence was proposed to increase resilience by training with a greater diversity of adversarial examples.

Defensive distillation is a defensive method developed by Papernot et al. (2016) to increase the resilience of Deep Neural Network (DNN) trained classifiers from adversarial attacks. Defensive distillation is based off the distillation method for neural networks produced by Hinton, Vinyals, and Dean (2014). Distillation is a method which was developed to reduce the amount of computational power required to perform predictive classification. Distillation works by training a DNN using additional information which is obtained from evaluating the DNN, to determine the classification probability of the training data to each class. The additional information is known as soft labels, which are used to label the data when training the distilled DNN. The soft labels reduce the amount of training points required and reduce the amount of bias in the network which can lead to overfitting. Defensive distillation uses the same approach as the original distillation, with the difference that the original network and the distilled network are identical in their architecture. The aim of defensive distillation is not to reduce the amount of training points but to utilise the soft labels to prevent adversarial examples from effectively poisoning the network.

An attack against defensive distillation was developed by Carlini and Wagner (2016), in which it was shown that defensive distillation was not as secure as originally thought. An extension of defensive distillation was developed by Papernot and McDaniel (2017) to increase the resilience in response to exposed vulnerabilities. The extension differs from the original in that additional information is used

in combination with the original classification probabilities. The additional information is a predictive uncertainty value which is computed from the original DNN.

2.7.2.3 On the (Statistical) Detection of Adversarial Examples

Grosse, Manoharan, Papernot, Backes, and McDaniel (2017) used statistical tests to identify adversarial examples from genuine data. The basis of the defence is that adversarial examples are not drawn from the same distribution as genuine data and can be identified through statistical tests. After identifying the adversarial examples, the ML model is modified with another class which is used to classify adversarial examples. The test used in the defence is a kernel-based two-sample statistical hypothesis test called the Maximum Mean Discrepancy (MMD) test, which was developed by Gretton, Borgwardt, Rasch, Schölkopf, and Smola (2012). The MMD test is as follows:

We test whether distributions p and q are different on the basis of samples drawn from each of them, by finding a well behaved (e.g., smooth) function which is large on the points drawn from p , and small (as negative as possible) on the points from q . We use as our test statistic the difference between the mean function values on the two samples; when this is large, the samples are likely from different distributions. (Gretton et al., 2012)

Although MMD was successful in detecting adversarial examples from genuine examples, Carlini and Wagner (2017) found that they were able to generate adversarial examples using the C and W attack algorithm which were not detected by the MMD defence.

2.7.2.4 Activation Clustering Defence

B. Chen et al. (2018) developed a defence for backdoor poisoning attacks on neural network models by analysing the neurons from the last hidden layer for anomalous clusters. The intuition behind the defence was that for the backdoor poisoning attack (where a trigger image was superimposed onto target images mislabelled with a target label, e.g., a stop sign with a trigger was labelled as a speed sign) is that the network learns both the features of the original source image and the target adversarial class (from the trigger). The authors proceeded by testing adversarial images and examining the neurons from the last hidden layer, which provided the required information to clearly separate the adversarial images from the clean. The results from performing the activation clustering defence on a poisoned MNIST dataset is shown in

.

Refer to
Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., . . . Srivastava, B. (2018).
Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. ArXiv eprints.
Retrieved from <https://arxiv.org/pdf/1811.03728.pdf>

2.7.2.5 Spectral Signatures

Tran, Li, and Madry (2018) identify a property of backdoor attacks which they name “Spectral Signatures”, they argue that backdoor attacks “tend to leave behind a detectable trace in the spectrum of the covariance of a feature representation learned by the neural network”. By identifying the spectral signatures residing in the network the authors were able to identify the poisoned inputs and remove them from the network. The process for identifying spectral signatures and removing the corresponding adversarial examples is described as follows:

We take a black-box neural network with some designated learned representation. This can typically be the representation from an autoencoder or a layer in a deep network that is believed to represent high level features. Then, we take the representation vectors for all inputs of each label. The intuition here is that if the set of inputs with a given label consists of both clean examples as well as corrupted examples from a different label set, the backdoor from the latter set will provide a strong signal in this representation for classification. As long as the signal is large in magnitude, we can detect it via singular value decomposition and remove the images that provide the signal. (Tran et al., 2018)

For the majority of the experiments, the authors were able to remove 100% of the adversarial examples by applying the spectral signature defence.

2.7.2.6 Neuron Pruning and Unlearning

Two defensive strategies were developed by Wang et al. (2019) to patch poisoned neural networks after identifying backdoor triggers within the dataset. The poisoning attacks were identified by examining the neuron activation weights from the last hidden layer of the model, as that layer contains all the encoded data from the previous layers. In the last hidden layer, the authors found that the

activation weights of adversarial examples were 3-7 times higher than clean input images. After identifying the model as being poisoned, an identification and filtering method is applied to the model to prevent any incoming adversarial examples from bypassing misclassification. While the filtering method is in place, the authors presented two defensive patches to fix the model.

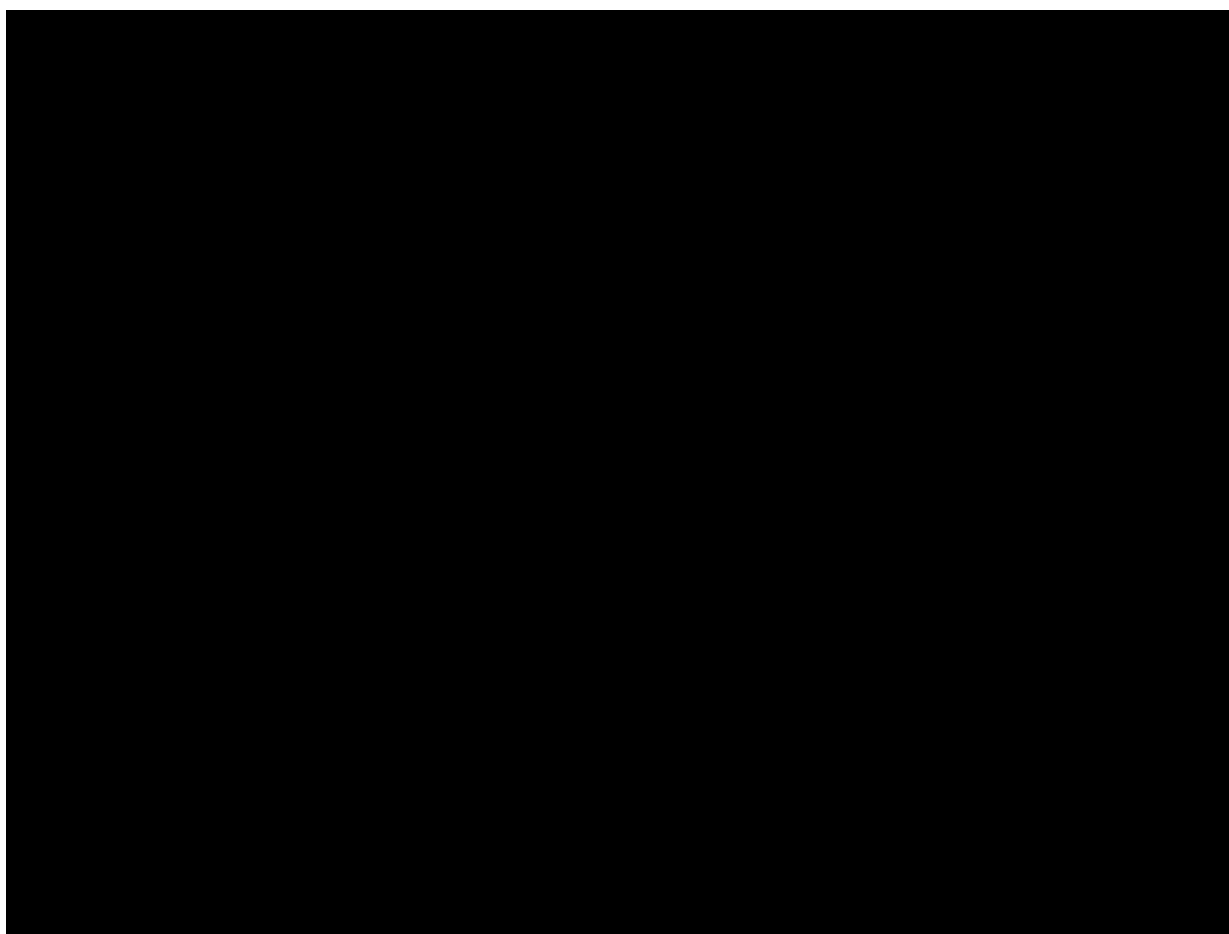
The first strategy is neuron pruning, it involves removing the backdoored neurons from the model to prevent the trigger from being activated, without significant negative impact to the general classification of the model. An example from one set of poisoned attacks had shown that only the top 1% of neurons were required to be active (i.e., all other neurons were masked to zero) for the adversarial example to be classified as its target label. Even though only 1% of neurons were required for the attack to work, in that example, 30% of the neurons needed to be pruned to reduce the success of attacks to near zero, this is due to neurons being trained with backup information for the model.

The second strategy is an unlearning approach, where the model is trained to unlearn the trigger. The unlearning is performed by reverse engineering the trojan trigger and examining how it affects the change in activation weight values of the network which can then be patched. The results from the patching experiments are shown in Table 8.

Table 8 - Model Patching Performance Comparison Adapted From (Wang et al., 2019)

Refer to

Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., & Zhao, B. Y. (2019, 19-23 May 2019). Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. IEEE Symposium on Security and Privacy (SP).<https://ieeexplore.ieee.org/abstract/document/8835365>



2.7.2.7 *Revisiting Adversarial Training*

Adversarial training is a robust algorithm approach to defend against adversarial examples by training the model to not only be accurate for the original task, but to be able to identify adversarial examples. The adversarial training process is performed by training the model on the original training data and with additional adversarial examples. According to Wong, Rice, and Kolter (2020) adversarial training was assumed to be quite computationally expensive when including the creation of the adversarial examples, but this is not the case in the experiments performed by Wong et al. (2020), as they were able to train a robust algorithm by using weaker adversarial examples which are computationally cheaper to generate using fast gradient sign method.

2.7.2.8 *Detection of Adversarial Training Examples in Poisoning Attacks through Anomaly Detection*

Paudice, Muñoz-González, Gyorgy, and Lupu (2018) developed a defensive strategy against adversarial poisoning attacks by pre-filtering the training data to remove adversarial examples by means of outlier detection. According to the authors, adversarial examples differ significantly from genuine training data points, which allows for their identification and removal. The defence is proposed as an alternative to testing each training sample to identify adversarial examples, as that approach, which can succeed in identifying poisoning attacks, is computationally expensive. The developed defence is for linear classifiers and is model agnostic. The defence was tested against two attacks, the optimal attack developed by the authors, and a label flipping attack. The label flipping attack was shown to be more resilient than the optimal attack to the authors defensive technique as the datapoints of the label flipping attacks are closer to the original data compared to the optimal attacks. The experiments were performed on the SpamBase and MNIST datasets. The optimal attack strategy increased the classification error of the SpamBase model from 0.112 ± 0.010 to 0.195 ± 0.019 and the MNIST from 0.037 ± 0.005 to 0.391 ± 0.160 . For the SpamBase model, the best performing defence achieved a 0.111 ± 0.009 classification error, and the worst performing defence achieved a 0.137 ± 0.015 classification error. For the MNIST model, the best performing defence achieved a 0.070 ± 0.013 classification error, and the worst performing defence achieved a 0.079 ± 0.022 classification error.

2.7.2.9 *Label Sanitisation against Label Flipping Poisoning Attacks*

Paudice, Muñoz-González, and Lupu (2018) developed a defensive strategy to defend against adversarial label flipping poisoning attacks, as the defence from the previously discussed paper did not perform as well against label flipping attacks compared to optimal attacks. The defence strategy utilises k-Nearest-Neighbours (k-NN) to identify adversarial data inputs which negatively affect the accuracy of the model. The defence was tested on linear classifier models which were trained on three datasets (UCI BreastCancer, MNIST and SpamBase). The defence was developed using a worst-case scenario where the attacker has complete white-box knowledge of the defender's system (e.g., model

architecture, training set, loss function). The authors understand that this is an unlikely scenario but wanted to test their defence under a worse-case scenario. The defence mitigates the label flipping attack by re-labelling identified outliers (suspicious data points) to malicious. The outliers are identified using k-NN to locate data points which are far from genuine points of the same label. The proposed defence was able to minimise the classification error from the label flipping attack. The results from the label flipping attacks and the sanitisation defence are shown in Figure 2.10, where the red line illustrates the performance of the model without a defence applied, and the blue line illustrates the performance after the defensive strategy was applied.

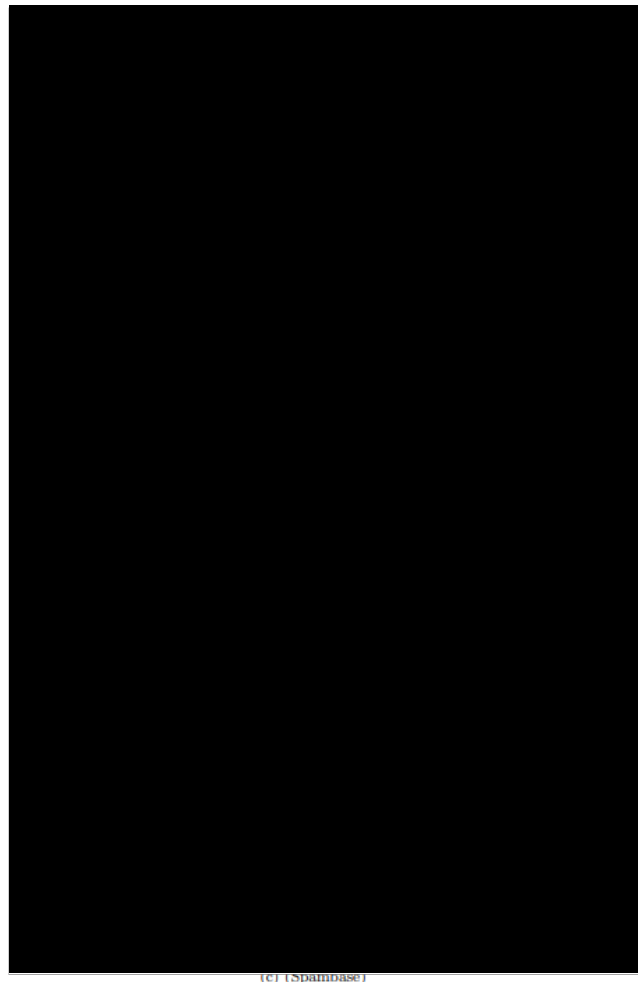


Figure 2.10 - Label Sanitisation Defence Results Adapted From (Paudice, Muñoz-González, & Lupu, 2018)
Refer to

Paudice, A., Muñoz-González, L., & Lupu, E. (2018). Label Sanitization against Label Flipping Poisoning Attacks. ArXiv e-prints. Retrieved from <https://arxiv.org/abs/1803.00992>

2.8 EMBER Dataset

The EMBER dataset is a collection of extracted features from 1.1 Million portable executable (PE) files (900,000 train and 200,000 test). The 900,000 training files are split equally into three categories, malicious, benign, and unknown. The unknown category is not used when training the ML model, so the actual size of the EMBER dataset when training is 800,000. EMBER was developed to serve as a benchmark for machine learning malware research.

In the EMBER dataset, the feature generation was performed from a selection of the PE32 file information shown in Table 9, which generates a 2351 bit dimension vector from each PE32 file using the feature hashing module from sklearn.

Table 9 - EMBER Feature Engineering

Feature Class	Vector Dimension Size	Information
ByteHistogram	256	Entire byte histogram
ByteEntropyHistogram	256	Histogram of joint probability of each byte value and local entropy
SectionInfo	255	Feature hash of section names, sizes, and entropy
ImportsInfo	1280	Feature hash of import libraries and their functions
ExportsInfo	128	Feature hash of exported functions
GeneralFileInfo	10	General file information (size, vsize, has_debug, exports, imports, has_relocations, has_resources, has_signature, has_tls, symbols)
HeaderFileInfo	62	Information about the OS, architecture, and other header information
StringExtractor	104	Extracted string information

As discussed in 2.3.1 Feature Engineering, the engineering of salient features is a crucial step in the development of an ML model. The features chosen for use by the researchers at Endgame are based upon the work of several researchers which have identified what features in a Windows PE32 file contribute the most when training a malware detection ML model. A summary of three feature engineering research papers, which the Endgame researchers followed in developing their feature engineering process are given below:

Paper 1 – Deep Neural Network Based Malware Detection Using Two-Dimensional Binary Program Features

Saxe and Berlin (2015) developed an approach for detecting malware files by using static features extracted from binary files to train a deep neural network. The feature engineering performed on the

binary files is comprised of extracting information from four sections of the binary file and creating a 1024-dimensional feature vector. The 1024-dimensional feature vector is comprised of four 256-dimensional feature vectors. The first set of feature vectors extracted from the binary file are the byte/entropy histogram of the file, which was performed to generate a model of the file's features in a file format agnostic approach. The second set of feature vectors extracted from the binary file are from the import address table. The import features 256 dimension vector is comprised of a hash of each tuple of import name and associated import function e.g. kernel32.dll:SetFilePointer. The imports section was included to identify the association of the external function calls the binary file relies upon to operate. The third set of feature vectors is a 256-bit hash of strings extracted from the binary file and the last set of feature vectors are computed from the numerical values extracted from the binary file's packaging, which were extracted using pefile.

Paper 2 – Data Mining Methods for Detection of New Malicious Executables

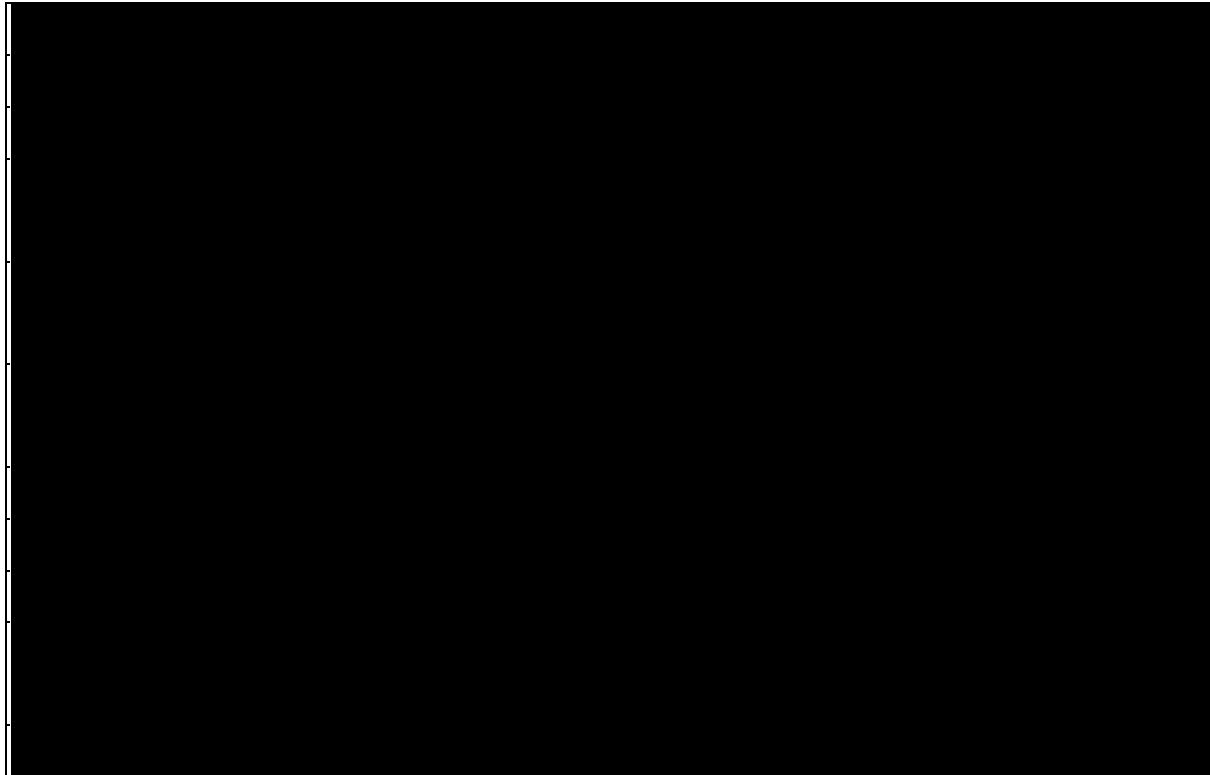
Schultz, Eskin, Zadok, and Stolfo (2001) developed a data-mining framework for identifying malicious files by identifying patterns which differentiate between malicious and benign files. The authors extracted static properties of the binary files to be used in generating a detection model to classify new unseen malicious files as malicious. The features extracted were from the system resource information, strings and byte sequences of the binary file. Three different learning algorithms were used to train the classification models. The first set of features extracted from the binary files to train the classification models were from the DLLs used, their function calls and the number of function calls within each DLL. The second set of features extracted were the strings contained within the binary file, and the last set of features extracted were the byte sequences of the binary file generated using hexdump.

Paper 3 – A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables

Shafiq, Tabish, Mirza, and Farooq (2009) developed a real-time PE-Miner framework which automatically extracts salient information from PE files to detect Zero-day malware files. The PE-Miner operates in three steps, the first is to identify which features are to be extracted from the PE file, the second is to perform pre-processing on the extracted features to remove redundant and irrelevant features, the third step is the selection of a data mining algorithm to perform the classification. The features the authors selected for extraction are shown in Table 10.

Table 10 - Realtime PE-Miner Extracted PE Features Adapted from (Shafiq et al., 2009)

Refer to Shafiq, M. Z., Tabish, S. M., Mirza, F., & Farooq, M. (2009, September). Pe-miner: Mining structural information to detect malicious executables in realtime. In *International workshop on recent advances in intrusion detection* (pp. 121-141). Springer, Berlin, Heidelberg. <https://link.springer.com/content/pdf/10.1007%2F978-3-642-04342-0.pdf>



The pre-processing was performed by using either Redundant Feature Removal (RFR), Principal Component Analysis (PCA) or Haar Wavelet Transform (HWT) and the classification was performed using a selection of five data mining algorithms: instance based learner (IBk), decision tree (J48), Naïve Bayes, inductive rule learner (RIPPER) and support vector machines. Shafiq et al. (2009) found that the decision tree model performed the best for the majority of experiments.

2.9 Summary

Machine learning is a subset of Artificial Intelligence which is utilised in a variety of different fields to increase productivity through training machines to automatically perform a task. Machine learning has been implemented in many different fields such as medical science, information technology, finance, and cyber security. Machine learning algorithms build models for the purpose of mapping an input to an output with a high degree of accuracy. These models significantly improve the performance of technology through automation of feature detection which previously required human input.

However, ML algorithms have shown to be susceptible to a variety of different adversarial attacks which target the integrity, availability, and privacy of ML models. The adversarial attacks are a threat to computer systems which rely on malware protection from applications which have been trained to detect malware through ML approaches. Critical Infrastructure (CI) systems are often not able to support the extended lengths of downtime required when updating and patching vulnerabilities and

instead rely on protection software to prevent vulnerabilities from being exploited. Machine learning is a new approach utilised in malware detection, which could be utilised for the detection of unknown-unknown (Zero-day) malware.

As ML-based malware detection applications require periodic retraining of new data to stay up-to-date in detecting new malware variants, this retraining provides an attack vector for adversaries to perform poisoning attacks, reducing the efficacy of the ML model. This research aims to explore the different avenues of adversarial attacks and to develop defensive strategies for protecting ML-based malware detection applications. A traditional scientific approach has been identified as the most appropriate method and the research has been designed with the risks, limitations and ethical considerations being examined.

Malware detection is an important area in cyber security. Computer systems around the world rely on malware detection applications to prevent malware attacks from succeeding. Malware detection is not a straightforward task, as new variants of malware are generated at an ever-increasing rate. ML has been utilised to generate predictive classification models to identify new malware which conventional malware detection methods may not detect. Machine learning however has been found to be vulnerable to different types of adversarial attacks, in which an attacker is able to negatively affect the integrity and availability of the ML model.

Different types of defences have been proposed to mitigate adversarial attacks, but no generally effective fix has been developed. For ML-based malware detection applications, an effective defence against adversarial attacks is key in ensuring the integrity of the application, as the open-training model used provides an attack vector for adversarial poisoning attacks. A summary of the problems identified are shown in

.

Table 11 - Summary of Identified Problems

Identified Problem	Addressed in Research	Related Research Question
Machine learning algorithms are vulnerable to adversarial attacks	Yes	RQ1
A general mitigation strategy has not been developed to counter all adversarial attacks	Yes	RQ2
Researchers do not share their malware databases, which makes replicating the results difficult	No	

The problems identified in the literature review and summarised in Table 11 only provide a broad overview of the issues relating to adversarial machine learning attack and defence research. An important problem which has not been covered in other research is the fact that for the majority of the developed adversarial machine learning defences to operate, they require a trusted clean dataset which can be used as a benchmark to identify and remove adversarial examples. The problem which arises from this method is that it is not possible to know for certain if the trusted clean dataset is in fact clean, or if adversarial examples have been introduced into the dataset without the researcher being aware. Trust in a clean dataset becomes more difficult when the data is required to be sourced online or from a third party, compared to a dataset which was curated inhouse. The problem with using an inhouse dataset, is that without the large variation in data which occurs when data is gathered from multiple channels, overfitting is more likely to occur in the trained ML model. The trust issue was addressed in this thesis when developing the defensive measure to prevent targeted adversarial poisoning attacks from succeeding.

3 Research Methods and Design

This chapter begins with a discussion of philosophical systems and research approaches. This is followed by a discussion of research methods and a justification of the selected method for this research. Next, the research design, describing the process used to develop the research questions and their corresponding hypotheses was done in accordance with the traditional scientific approach, which was identified as the most appropriate research approach for this thesis. Next, an outline of the research phases was produced, starting with the acquisition and analysis of malware databases, the development of ML models, the development and execution of the preliminary and main experiments, and finishing with the development of a defensive strategy to mitigate the targeted adversarial poisoning attack. The chapter concludes by covering the materials required to complete the research phases, an overview of the risks and limitations and ethical considerations related to the research topic are discussed.

3.1 Philosophical Systems

The philosophical systems which encompass scientific research are ontology, epistemology, methodology, and methods (Zukauskas, Vveinhardt, & Andriukaitienė, 2018). These four areas provide the basis for how new information is obtained through a research process. Different researchers have developed different systems which encompass the four areas and provide a variety of approaches for performing scientific research. In

, an overview is given of a few of the different systems. The areas described in

were explored and a number of methodologies were considered, but all were ruled out except for the traditional quantitative scientific approach of hypothesis development and experimentation.

Table 12 - Research Methods Adapted from Williamson and Johanson (2017)

Research paradigm (world view)	Ontology	Epistemology	Research methodology	Modes of inquiry
Interpretive	Society is constructed and social reality is constantly interpreted	Knowledge is subjective and is generated through “exploration of the beliefs, feelings and interpretations of research participants” (Williamson & Johanson, 2013)	Qualitative	Interviews Case studies Field Experiment
Positivist	A theory, if not proven false, is corroborated but not proven true	Knowledge is objective and generated through observation.	Quantitative	Experimental Quasi-Experimental Surveys

3.1.1 Scientific Research Paradigms

A paradigm refers to “a system of ideas, or world view, used by a community of researchers to generate knowledge. It is a set of assumptions, research strategies and criteria for rigour that are shared, even taken for granted by that community” (Fossey, Harvey, McDermott, and Davidson, 2002, p. 718).

There are currently three main philosophical paradigms currently accepted by the scientific community as being the most appropriate in exploring the nature of reality, these systems are positivism, constructivism, and interpretivism. Using any of these philosophical approaches enables researchers to assess their research in either objective or subjective ways. Each of these scientific paradigms were explored, but as constructivism and interpretivism were not appropriate choices for the research performed in this thesis, they will not be discussed in the following sections.

3.1.1.1 Positivism

The positivist paradigm is focused on the objective analysis of data. It does not take into consideration the feelings of the researcher when examining the research to identify the truth (Guba, 1981). Positivism is strongly associated with quantitative research methods, analysing numerical data obtained through traditional scientific experiments (Zukauskas et al., 2018). The positivist research process is illustrated in Figure 3.1, it starts with the development of hypotheses which can be empirically tested and ends with the writing of results. The positivist researcher aims to gather knowledge through empirical testing which support their hypotheses and discover a set of general laws (Neuman, 2014).

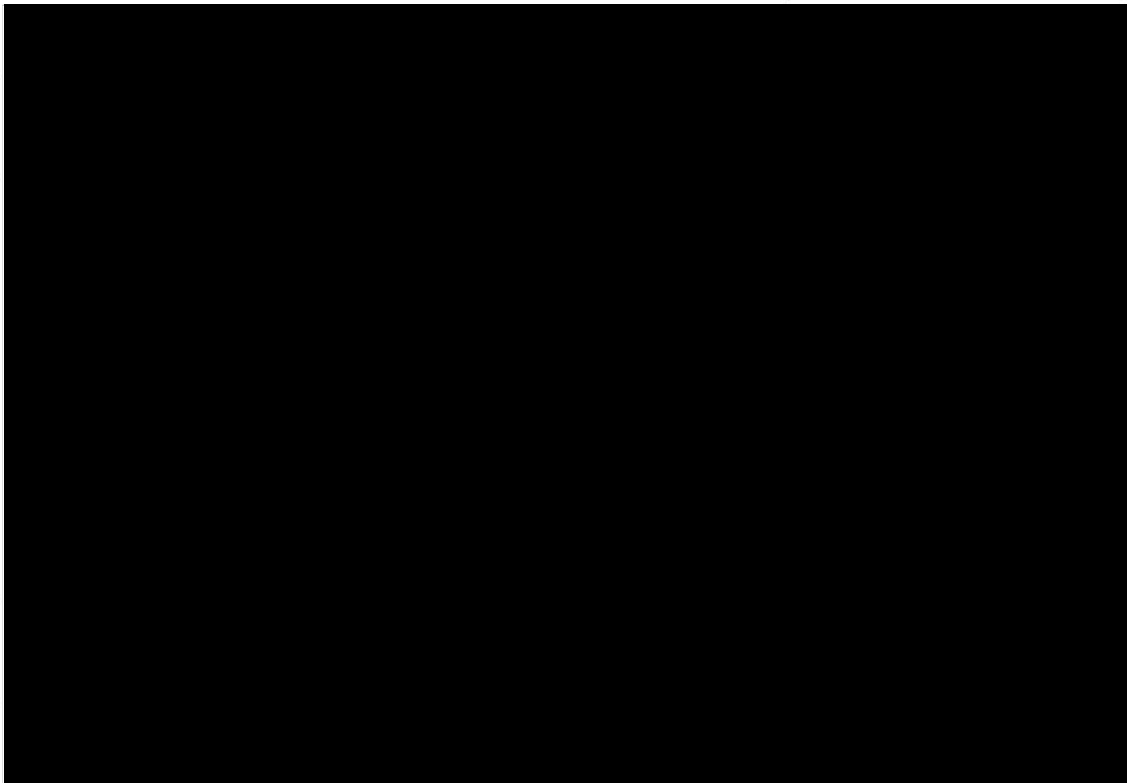


Figure 3.1 - Positivist Research Process Adapted from (Williamson & Johanson, 2017)

3.1.1.2 Existing Taxonomies of Research Approaches

Galliers (1990) identified several different research approaches in the domain of information systems, as shown in Table 13. The approaches identified are categorised across a spectrum ranging between qualitative and quantitative. Each approach identified has been analysed to determine which type of research would be most suitable for the research described in this thesis. The acceptable approaches were narrowed down to the traditional empirical approaches (quantitative), as performing experiments and analysing the results is a core aspect of the proposed research. The approaches chosen for consideration are laboratory experiment, field experiment, theorem proof and simulation.

Table 13 - Information Systems Research Approaches. Adapted from (Galliers, 1990)

Refer to

Galliers, R. D. (1990). Choosing appropriate information systems research approaches: a revised taxonomy. In *Proceedings of the IFIP TC8 WG8. 2.*

Theory Building	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Theory Testing	Yes	Yes	Yes	Possibly	Possibly	No	Possibly	No	No	Possibly
Theory Extension	Possibly	Possibly	Possibly	Possibly	Possibly	No	No	No	No	Possibly

A laboratory experiment is the most suitable approach for research which requires high precision in controlling the variables and environment to produce an experiment which can be easily reproduced to replicate the findings. Due to the nature of the research questions involving malware and the need for a high level of control, a laboratory experiment is a suitable research approach for the development, testing and analysis required to obtain the answers from the proposed research questions.

A field experiment is not as suitable as a laboratory experiment due to the malware dataset required for experimentation. If a field experiment was chosen, a honeypot environment would be required for the collection of new malware samples, in addition to the pre-assembled malware database. The timeframe of collecting wild malware would be unknown, which would not be suitable, in addition to the fact that it would not be known if the captured files are malicious in every instance and controlling the families which the captured malware belong to would also be difficult. Due to the aforementioned issues, a laboratory experiment is more suitable, with field experiments being left to future research.

Theorem proof as defined by Vogel and Wetherbe (1984) are “application areas from fields such as Computer Science that otherwise would not be identified”. The aim of the research is not to develop an ML algorithm but instead, perform an evaluation of the different family of algorithms and develop an adversarial attack/defence. As the evaluation better fits the laboratory experiment approach, theorem proof was not selected for use.

Simulation is a suitable approach for research problems where undertaking a field experiment is not appropriate, but a simulation of a live environment can be developed from observing a live scenario and generating the required variables, to replicate the environment as accurately as possible. A simulation approach does not provide any added benefits over a laboratory experiment for the proposed research. The generation of adversarial malware examples is needed to be done in a

laboratory setting, as a high level of precision is needed to control how the adversarial attacks perform and to provide control for replicating the experiments.

Edgar and Manz (2017) suggest a decision tree approach for determining the most appropriate research method. The authors go into detail for each of the different research approaches which are applicable for the many different sub-fields of cyber security. The main overarching research fields identified are observational research, mathematical research, experimental research and applied research. An illustration of the decision tree is shown in Figure 3.2.

Observational research methods cover open-ended and broad research topics. The methods are most suitable for research topics which aim to understand a cyber system, without some preconceived hypothesis of expected behaviour. Mathematical research methods cover the theoretical approaches previously defined by Vogel and Wetherbe (1984) and the simulation approach presented by (Galliers, 1990). Experimental research methods cover Hypothetico-deductive research and Quasi-experimental research. Hypothetico-deductive research is the term used by Edgar and Manz (2017) to refer to the traditional scientific approaches such as the field and laboratory experiment approaches proposed by Galliers. Quasi-experimental research approaches are similar to the traditional experimental approach but differ in that control of the variables is difficult to obtain, which leads to less reliable results. Applied research methods cover applied experiment approach and applied observational study. Applied research methods are suitable for research which aims to evaluate the efficacy of a system for solving a problem.

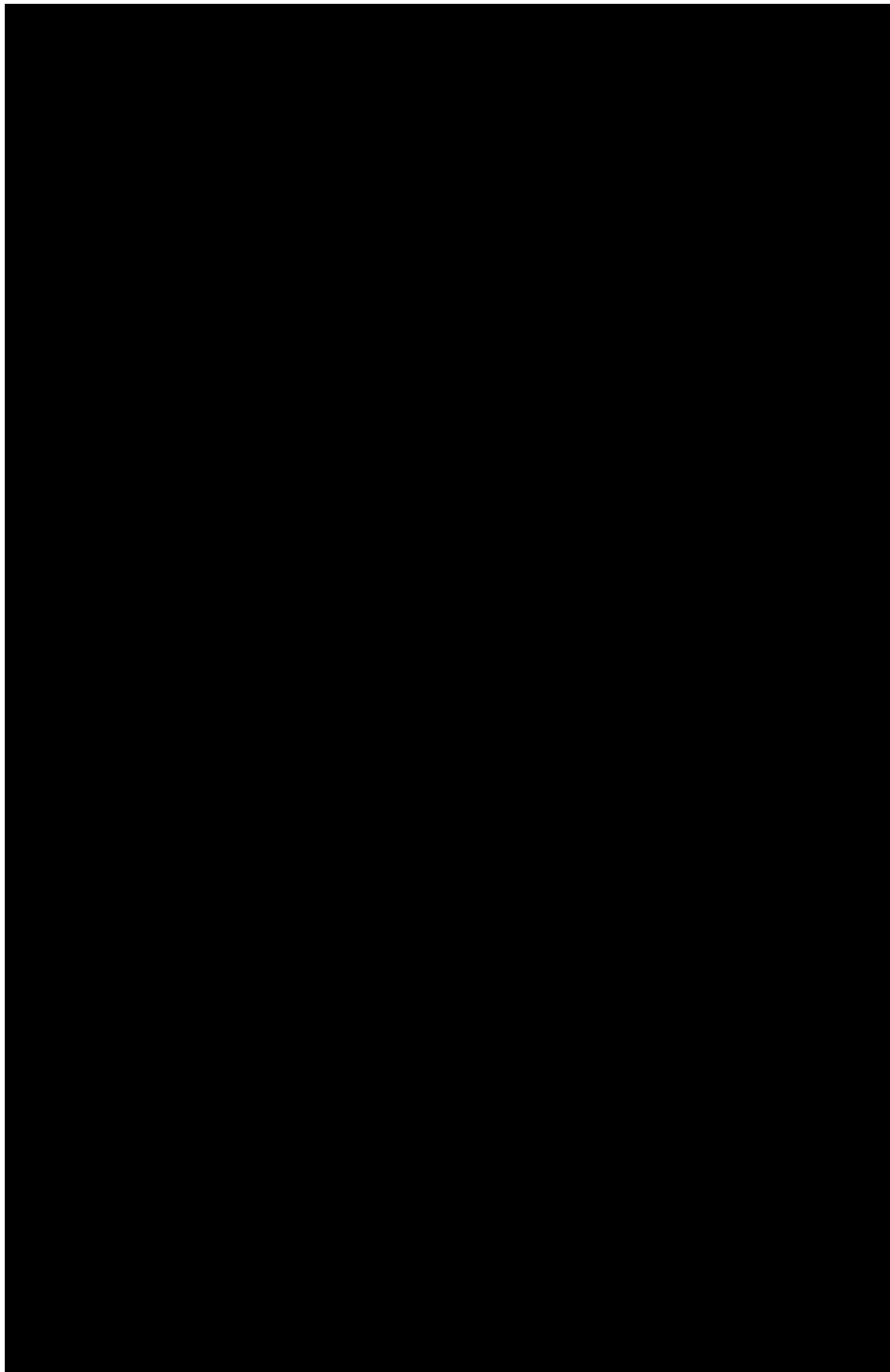


Figure 3.2 -Edgar and Manz Research Decision Tree Pt.1 Adapted from (Edgar & Manz, 2017)

Refer to

*Edgar, T. W., & Manz, D. O. (2017). Research Methods for Cyber Security: Syngress Publishing.
ESET. (2020). Trojan Horse. Ch.3, P.80. Retrieved from <https://www.eset.com/uk/types-of-cyber-threats/trojanhorse/>.*

3.1.1.3 Cyber Security Research

Edgar and Manz (2017) proposed a decision tree structure to identify which research method is most suitable for a proposed research problem. There are four overarching categories of research from which a cyber security researcher can select. The four categories of research are Theoretical, Observational, Experimental, and Applied Research, as described by Edgar and Manz (2017) below.

Theoretical research is the examination of the behaviour of a cyber system. It involves theorising or defining the operation of a cyber system and its environment. The two methodologies of theoretical cyber research are Formal Theory and Simulation. Formal theory is the development of mathematical proofs and internal validity. Cryptographic research is a research field which is strongly associated with formal theory research. Simulation theory is used in areas which contain complex problems that are difficult to develop a formal model for. Simulation theory is used to test smaller sets of data to identify if the area is worth pursuing at a more complex level.

Observational research is appropriate when the research question involves the understanding of a real-world cyber system. The methods used in observational research are Exploratory and Descriptive Studies. Exploratory and descriptive studies both involve the collection, analysis, and interpretation of data. The two methods differ in their scope, with exploratory studies being more general and examining entire systems, and descriptive studies being more in depth on a particular aspect of a system.

Edgar and Manz (2017) include ML in their taxonomy of research methods. This is perhaps unusual, but in their view, ML as a research method is for automating phases of the research cycle. Machine learning as a research method does not focus on the use of ML when undertaking research, such as developing an algorithm or defence for an adversarial attack. The use of machine learning as a research method was defended by Edgar and Manz (2017) as an approach due to the significant progress in machine learning research and its application in the cyber security domain.

Experimental research is the traditional experimental process which is most commonly associated with scientific research. The methods used in experimental research are Hypothetico-Deductive and Quasi-Experiment. Hypothetico-deductive is the term the authors use for the traditional scientific experiment approach, where a hypothesis is developed, and experiments are performed to gather information to either support the hypothesis/null-hypothesis. The quasi-deductive approach is similar to the hypothetical-deductive approach but differs in the control of the experiment. If a researcher is unable to perform their experiments in a real-world scenario but can create a simulation with control over the variables, then a quasi-experiment is the appropriate method of choice.

Applied research is the process of identifying how well the information gathered from other experiments is used to solve a given problem. The methods used in applied research are applied experimentation and applied observational study. Applied experimentation identifies how well a proposed solution is by evaluating a set of controlled experiments. The results from the experiments are used as a benchmark for comparison from the original operation of the cyber system and to any new solutions which may be developed. Applied observation study is the evaluation of cyber systems in different situations testing a variety of conditions for the operation of the system. It is a useful method for identifying the bounds within which a cyber system is able to operate.

From following the decision tree developed by Edgar and Manz (2017), the hypothetico-deductive research method was determined to be the most appropriate choice. This outcome is consistent with the result using the taxonomy of Galliers (1990), that a traditional scientific experiment approach is the most suitable for conducting the proposed research.

3.2 Research Design

The research design is divided into four sections, research questions, hypotheses, research phases, and materials. Each section covers an area required for the development of the experiments when undertaking a traditional scientific research approach, as it was identified in the previous section as the most appropriate research approach for the topic of this thesis. The research questions and hypotheses sections are straight forward, they contain the proposed research questions and their related hypotheses which were tested to determine if it was possible to induce targeted false negatives in ML malware detection applications and if it is possible to detect the false negatives. The research phases section outlines the path the research process followed, starting from acquiring the relevant databases and concluding with the documenting of analysed results. The materials section outlines what was required to setup and undertake the experiment phases.

3.2.1 Hypotheses

From the proposed research questions stated in section 1.4, the following hypotheses were developed and tested throughout the experimentation phase of the research.

H₁. A manual selection of features can be used for a successful adversarial poisoning attack.

H₂. A random selection of features can be used for a successful adversarial poisoning attack.

H₃. No more than 5% of the benign training data feature space is required to be poisoned to reduce the general efficacy of the model.

H₄. No more than 10% of the benign training data files are required to be poisoned to reduce the general efficacy of the model.

H₅. No more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.

H₆. The targeted adversarial attack can be prevented at test time.

The research questions and their corresponding hypotheses are illustrated in Table 14.

.

Table 14 - Research Question Relationships

Research Question	Hypotheses	Output
RQ1	H ₁ , H ₂ , H ₃ , H ₄ , H ₅	Identify if a targeted adversarial attack can be successfully performed
RQ2	H ₆	Develop defensive strategy for targeted adversarial poisoning attack

3.2.2 Research Phases

3.2.2.1 Acquire Datasets

The initial phase of research was obtaining the required Microsoft Windows PE file datasets for development of a machine learning based malware detector and to be used as the samples for creation of adversarial malware. The datasets were acquired from the anti-malware vendor VirusShare and from anti-malware researchers at Endgame. As VirusShare contained a significant amount of malware samples and the EMBER dataset contained 1.1 million data samples, no further datasets were sought for use.

3.2.2.2 Perform Feature Analysis

The second phase was to perform feature analysis on the datasets. The EMBER dataset provided a script for performing feature engineering on new data in the same format as EMBER, this allowed for the binary malware files obtained from VirusShare to be converted into the JSON format which EMBER used.

3.2.2.3 Test Malware Detection Model

The third phase was to evaluate the performance of the ML malware detection model provided in the EMBER dataset on the selection of malware files obtained from VirusShare. It was required to identify which malware files from VirusShare could be possible candidates for the targeted adversarial poisoning attacks. The EMBER model is a GBDT which uses the default LightGBM parameters. EMBER uses the AUC ROC curve method for comparing the performance of binary classifiers. The default EMBER model achieves an AUC ROC score of 0.9991123, which is used as a baseline for comparison in the adversarial poisoning attack experiments. The same parameters are used to train the GBDT models in the general efficacy and targeted adversarial poisoning attacks. The code for training the GBDT EMBER model is in Appendix A – Machine Learning Model Code.

3.2.2.4 Test General efficacy Attack

The fourth phase was to perform an adversarial poisoning attack with the aim of reducing the general efficacy of the EMBER model. The aim was to identify if it is possible for the features from the selected malware files, which will act as Zero-day malware files, are acceptable for use in the targeted adversarial poisoning attack. Different sets of features and different injection percentages were tested to provide the targeted adversarial attack with a set of features to choose from. The features were selected from the imports section of the malware files and the poisoning attacks were performed in steps of 5 percent. The imports section was selected as inserting import functions into the EMBER JSON files was a straightforward task and the imports section has been identified by other researchers as a positive contributing feature space for malware detection (section 2.8)

3.2.2.5 Test Targeted Adversarial Attack

The fifth phase was the development of a targeted adversarial poisoning attack which allows for the targeted Zero-day malware file to bypass detection, while the general efficacy of the model is not significantly affected. The attack was different from the current adversarial poisoning attacks performed against malware detection models e.g., label flipping. Testing with different sets of features and at different poisoning percentages was performed to identify the optimal parameters for performing a successful targeted poisoning attack. The model performance was compared using the AUC ROC and the threshold value to determine if the targeted adversarial poisoning attacks were capable of allowing for a targeted Zero-day malware file to bypass detection, while the general efficacy of the model was within an acceptable variation from the baseline metric of the default clean EMBER model.

3.2.2.6 Test Different Machine Learning Model

The sixth phase was to replicate the previous attacks on an MLP model trained using a subset of the data from the EMBER dataset. A smaller dataset was used in the MLP model as the workstation was not powerful enough to train an MLP model using the entire EMBER dataset. The results from which were used to identify which model was the focus of the defensive strategy. The AUC ROC and threshold scores are used to compare the efficacy of the trained MLP models as they were used in the previous GBDT training/testing phases. A direct comparison of performance between the GBDT and MLP models are discussed in section 5. The MLP model was configured as a feed forward dense model with the following parameters:

- RELU Activation at hidden layers
- Sigmoid Activation at output layer
- Five hidden layers of [2048, 1024, 512, 256, 10] neurons
- Input_dropout=0.05
- Hidden_dropout=0.1

- Batch_size=128
- Epochs=100

In a feed forward dense ML model, each neuron is connected to all the neurons in the following layer and the only direction of the connections is forward. The code for the MLP model is in Appendix A – Machine Learning Model Code.

3.2.2.7 Develop/Test Defensive strategy

The seventh phase was to develop a new defensive strategy to defend against the targeted adversarial attack. The defensive strategy was developed to focus on identifying suspicious files at test time, in contrast to the current defences which identify poisoning at the training stage. The defensive strategy was developed under the assumption that it is not possible to obtain a trusted clean dataset to develop a baseline for the identification and removal of adversarial features from the training dataset.

3.2.2.8 Document Results

The eighth phase was the final documentation after analysing the results from the adversarial attack and defence phases. An analysis of the other defences and a comparison with the proposed defensive strategy was also provided

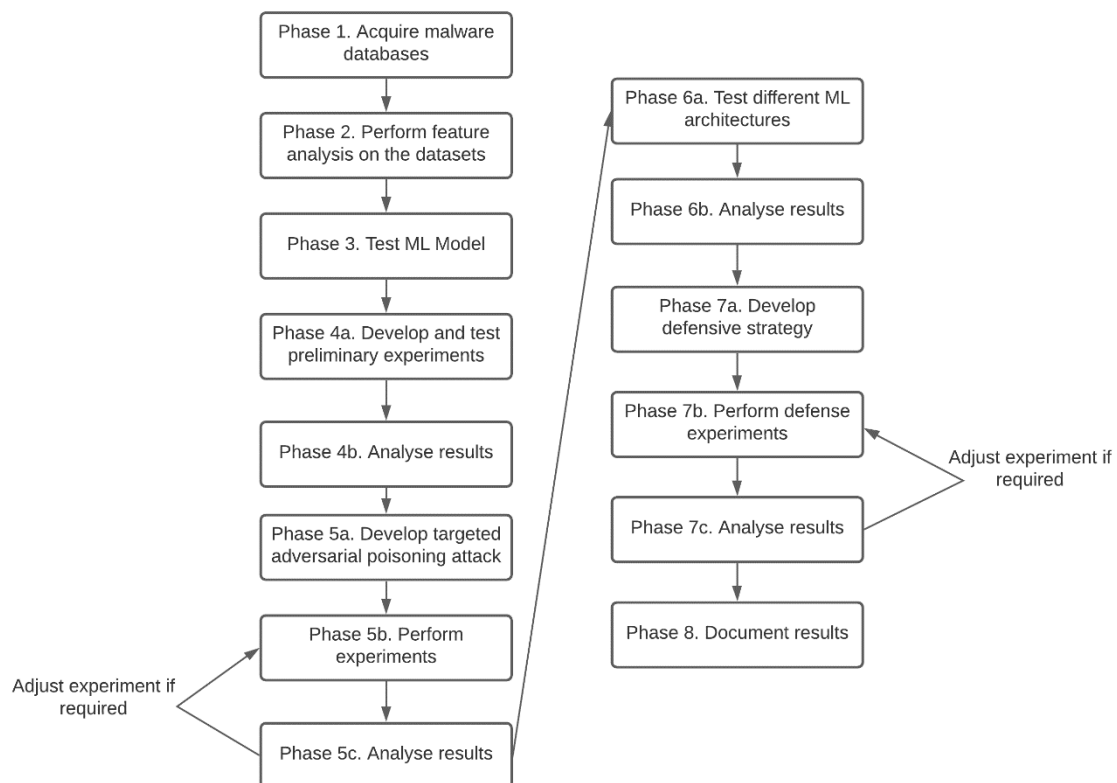


Figure 3.3 - Research Design

3.2.3 Materials

- Workstation (Detailed in Section 4.1)
- Malware Database(s) (Detailed in Section 4.2)
- Machine Learning Toolkit(s)

A workstation was required to develop the laboratory environment which was comprised of virtual machines in a segregated network, as the experimentation was on files of a malicious nature. Multiple malware databases were obtained to generate the malware detection model and the adversarial examples.

3.3 Risks and Limitations

The research was limited in selecting a traditional experiment approach over a field experiment in that the Zero-day malware selected for testing was simulated by using a malware obtained after a cut-off date from the training dataset, instead of being obtained through a honeypot or waiting for a new Zero-day malware to be identified. The limitation is not severe, as all of the malware sourced is real, and the experiments represent a real-world scenario of ML malware detection development. Other risks which have been identified are shown in Table 15, with their corresponding mitigation strategy and likelihood of occurrence.

Table 15 - Risks and Mitigation Strategies

Risk	Mitigation	Likelihood/Severity
Accidentally exposing malware to a network.	Experimentation will be performed on a private network. If for some reason a private network is not suitable for an experiment, appropriate measures will be taken to ensure it is not exposed to a live network.	Low
No access to proprietary malware detection models	Develop substitution environments where needed using published detection models	High
ML algorithm does not perform as intended	Choose another algorithm for the potential candidate selection	Moderate
Access to Zero-day malware	Generate new malware from old samples	Low
Loss of material	Perform backups	Moderate
Malware database stolen	Encrypt data	Moderate

3.4 Ethics

The research topic was on the efficacy of different machine learning algorithms in the application of malware detection and the resilience of the algorithms to adversarial machine learning attacks. The research did not involve any living creature. The experimental data are not of a private or sensitive nature. The results of the research have not in any way contributed to oppression or humiliation of any group or individual.

The ethics declaration was approved on 22/11/2018 under declaration number 22307.

3.5 Summary

In this chapter a description of the different scientific paradigms was presented along with the approaches that researchers utilise in the development of their research design. From examining the different scientific paradigms and the research approaches available, it was determined that the traditional scientific approach of hypotheses development and testing was the most appropriate approach for exploring the research questions proposed in this thesis. A selection of research phases were developed to frame the experimentation in a logical sequence of events, starting with the acquisition of malware datasets and finishing with the analysis of the experiments' results. A description of the materials required to undertake the experiments was given, along with an overview of the risks and limitations identified which could impact the research. Finally, the ethical considerations of the research topic were stated.

In the following chapter, the environment required to perform the experiments is discussed, along with an analysis of the datasets acquired for experimentation. The chapter follows with the preliminary and main experiment sections. In the preliminary experiment section, a selection of candidate adversarial features are evaluated in a variety of adversarial poisoning attacks to determine which set of adversarial features are to be used in the targeted poisoning attack in the main experiments. In the main experiment section, a set of targeted adversarial poisoning attacks are performed at varying degrees of injection against both GBDT and ANN trained ML models. Finally, a defensive strategy for mitigating the targeted adversarial poisoning attack was proposed, which identified false negatives at test time.

4 Experimentation – Adversarial Poisoning Attacks and Defences

In this chapter, research phases one through to seven are documented. The first two phases cover the acquisition of malware datasets from EMBER and VirusShare and the feature analyse process which was performed. In the third phase the malware files obtained from VirusShare were scanned using the GBDT model generated from the EMBER dataset to identify malware candidates for the experiment phases. The fourth phase documents the testing of the preliminary experiments which identified that it is possible to reduce the overall efficacy of trained EMBER ML models through indiscriminate poisoning attacks. The fifth phase documents the process for the development and testing of a targeted adversarial poisoning attack, which, following the process explored in phase five, poisons the EMBER dataset but does not reduce the overall efficacy while inducing a targeted false negative. The sixth phase introduces the use of another ML algorithm (ANN) to test the targeted adversarial poisoning attack and compares the results from the default EMBER GBDT model and the new ANN model. The seventh phase documents the defensive strategy which was developed to identify false negatives at test time.

4.1 Environment

All experiments were performed on a desktop PC with the following specifications:

- OS - Ubuntu 18.04
- CPU - AMD Ryzen 7 3700X 8-Core Processor
- RAM - 32GB DDR4 C16 3200Mhz
- GPU – AMD RX5700
- SSD – 512GB Samsung EVO 860

At the time of experimentation, the RX5700 did not have ROCm support for Keras. ROCm is an open-source development platform for AMD GPUs to accelerate compute-intensive tasks such as ML. The implications for this research were, that without this support, ML training tasks would take some time.

4.2 Datasets

To obtain the malicious files, a request was sent to both VirusShare and VirusTotal to gain researcher access to the data repositories. Access to VirusShare was granted but access to VirusTotal was not obtained. There may have been a problem with the VirusTotal application process, as no confirmation email was received. As access to VirusShare was granted, gaining further access to VirusTotal was postponed until a later date if required. The VirusShare files which were chosen as the Zero-day malware in this research were analysed by VirusTotal through their open API, this was done to

identify if any of the chosen VirusShare files had been submitted and identified as malicious before the cut-off date of the EMBER dataset (2017). None of the VirusShare files had been identified as malicious during or before 2017, and as such, were acceptable for use as Zero-day malware.

4.2.1 EMBER

The dataset chosen for testing adversarial poisoning attacks and defences was the Endgame Malware BEenchmark for Research (EMBER) dataset. The EMBER dataset is a collection of extracted features from 1.1 Million portable executable (PE) files (900,000 train and 200,000 test). The 900,000 training files are split equally into three categories, malicious, benign, and unknown. The unknown category is not used when training the ML model, so the actual size of the EMBER dataset when training is 800,000. EMBER was developed to serve as a benchmark for machine learning malware research. At the time of writing, there are three available EMBER datasets, EMBER2017, EMBER2017_v2, and EMBER2018. Both EMBER2017 and EMBER2017_v2 contain PE files scanned no later than 2017, and EMBER2018 contains PE files scanned no later than 2018. The features from the PE files were extracted using LIEF (Library to Instrument Executable Formats) and saved into JSON format. LIEF is a cross-platform library designed to parse, modify, and abstract various binary executable formats, including Windows PE files.

4.2.1.1 EMBER Model Performance

The third phase was to train the ML models which was done in two stages, training the clean models and training the poisoned models. The clean models are required to be used as a baseline for evaluating the efficacy of the poisoning attacks and defences. A clean trained GBDT model is provided along with the EMBER datasets and a summary of each model's performance is shown in Table 16 and Table 17 for 1% and 0.1% FPR. The performance of each model is calculated by the set FPRs (1% and 0.1%), the threshold value separates the benign and malicious files for the different FPRs. As an example, the EMBER2017 model at 1% FPR classes a file as malicious if it's predicted score is above 0.529.

Table 16 - EMBER Performance 1% FPR

Model	ROC AUC	Threshold	FNR	Detection Rate
EMBER2017	99.911%	0.529	1.838%	98.162%
EMBER2017_v2	99.908%	0.541	1.781%	98.219%
EMBER2018	98.495%	0.850	17.071%	82.929%

Table 17 - EMBER Performance 0.1% FPR

Model	ROC AUC	Threshold	FNR	Detection Rate
EMBER2017	99.911%	0.871	7.009%	92.991%
EMBER2017_v2	99.908%	0.884	7.671%	92.329%
EMBER2018	98.495%	0.951	35.794%	64.206%

4.2.2 VirusShare

VirusShare, the online virus repository, was used to obtain the malicious files required for testing the proposed adversarial poisoning attack. From VirusShare, the pack 00352 was selected for use because it was released at a later date than the EMBER2017 dataset, which was, at the time of exploratory examination, the only EMBER dataset available. After obtaining the malicious file package from VirusShare, each file was analysed by VirusTotal to identify if any of the files had in fact been submitted during 2017 or before. The VirusShare_00352 package contained 65,536 malicious files, which were scanned using the clean EMBER2017 model. Out of the 65,536 files, 2555 were PE32 files, which was determined by examining the contents of each file for the digital signature “4d 5a”. From the 2555 PE32 files, 2040 were correctly classified as malicious using the EMBER2017 model. The VirusShare package contains malicious files in general, it was not a package that only contained files of a certain nature e.g., Trojan or Ransomware.

The EMBER2017_v2 and EMBER2018 models were also used to scan the 2555 PE32 files. As the two models were generated using training data from 2018, unlike EMBER2017 which consisted of only files up to 2017, it was expected that the models would have a better classification rate as they were trained on newer data. An increase in the successful classification of malicious files would indicate that continuous retraining using newer files is required for the successful classification of Zero-day malware. This continuous retraining of models is what provides an attack vector for a malicious actor to perform an adversarial poisoning attack.

Out of the 2555 PE32 files, EMBER2017_v2 classified 2014 as malicious while EMBER2018 classified 1753. The Endgame researchers did state that EMBER2018 was constructed in such a way that it would be harder for the machine learning algorithms to classify accurately. The EMBER2017_v2 model, which was trained using files released up to 2018, incorrectly classified 26 more files (around 1%) than the EMBER2017 model, which was trained on files only from 2017 and earlier.

4.3 Feature Analysis

From the eight feature classes, the ImportsInfo class was chosen as the target for the adversarial poisoning attack. The import section of PE files contains a significant number of variables to choose

from for poisoning. Injecting a selection of import features into the training data can be easily performed by appending the feature to the correct section of the JSON file. The ImportsInfo class comprise the majority of the 2351 feature vectors (54%). Saxe and Berlin (2015) found that the import section by itself was the worst performing area for detecting malicious files but despite this, the ImportsInfo class will be evaluated for its adversarial poisoning performance.

The feature analysis of the EMBER2017 and VirusShare_00352 datasets began with counting the import libraries and their import functions. From the EMBER2017 dataset, the 900,000 samples from the training set were analysed to identify the most common unique libraries and the most common import functions from each of the labelled classes (benign, malicious and unknown). From the analysis, five unique libraries from the top ten were selected for the poisoning attacks. Five libraries were chosen as selecting more libraries would reduce the amount of target Zero-day files which contained all libraries, and the number of benign files which could be poisoned would also be reduced. From each of the unique libraries, different selections of import functions were chosen for the poisoning attacks. In the first test, five and ten import features which were found to be more common among the malicious data were chosen for injection. In the second test, five and ten features were randomly selected from the top one hundred import features from the malicious data. The chosen unique libraries and their total count are shown in Table 18.

Table 18 - Import Library and Function Amount

Library	Class	Import Count	Library	Import Function Count
Kernel32.dll	Benign	153,370		9,756,993
	Malicious	242,290		9,118,519
	Unknown	219,259		12,524,850
Shell32.dll	Benign	41,410		241,822
	Malicious	92,838		271,802
	Unknown	93,187		339,111
User32.dll	Benign	84,533		4,598,947
	Malicious	154,733		3,476,420
	Unknown	150,828		6,521,411
Advapi32.dll	Benign	84,554		1,118,428
	Malicious	100,853		859,340
	Unknown	124,522		1,284,236
Msvcrt.dll	Benign	33,356		1,068,236
	Malicious	61,809		1,629,194
	Unknown	43,510		1,218,773
All Libraries	Benign	1,394,044		39,953,980
	Malicious	1,226,244		21,911,172
	Unknown	1,573,555		36,579,239
	Total	4,193,843		98,444,391

4.4 Preliminary Experiments

As stated in Section 3.2.2, before performing the targeted adversarial poisoning attacks the fourth project phase was to conduct preliminary experiments to examine the efficacy of injecting import features into the benign feature space as an indiscriminate poisoning attack. The aim of the preliminary experiments was to identify the parameters for the targeted poisoning attacks i.e., should the import features chosen for poisoning be randomly or manually selected, what percentage of injection should be performed, should the injection percentage be based on the total feature space or a percentage of the training data amount?

The preliminary experiments were split into two categories, manual import function selection and random import function selection. Each category contained two sets of preliminary experiments, which tested adversarial poisoning attacks at different percentages of injection. Two methods were used to calculate the poisoning percentages. The first method calculated the poisoning percentage from the total number of import functions from all import libraries in the benign training files, which is referred to as the total benign feature space. The first set of preliminary experiments were designed to test H_3 , in that no more than 5% of the total benign feature space was to be poisoned. After examining the logs and identifying that for each poisoning experiment, 100% of the benign training files which contained the target import library were injected with the adversarial features, the method for calculating the poisoning percentage for the second set of preliminary experiments was altered.

In the second set of preliminary experiments, the poisoning percentage was calculated from the number of benign files which contained the target import library. This method is referred to as the target benign files. The second set of preliminary experiments tested poisoning in steps of 5% of the target benign files, starting at 5% and ending at 100%, with the 100% attack being the same as the attack in the first set of preliminary experiments. The second set of preliminary experiments were developed to identify at what percentage of poisoning of the target benign files does the quality of the model start to degrade.

The import functions selected came from a pool of the total import functions extracted from the malicious files in VirusShare_00352 which were identified as malicious by the EMBER2017 model. The total amount of import functions is shown in Table 19 and the top 20 import libraries are shown in Table 20. All the ML models generated in the preliminary attacks used the code to generate a GBDT model which was supplied with the EMBER dataset. The EMBER GBDT code was used as it provided a clear comparison to the clean EMBER model.

Table 19 - VirusShare_00352 Import Library and Function Count

Import Library	Import Count	Function Count
kernel32.dll	2125	182133
shell32.dll	1115	4398
user32.dll	1560	111484
advapi32.dll	1412	13767
msvcrt.dll	128	2007

Table 20 - Top 20 Import Libraries

Position	Import Library	Total Count	Position	Import Library	Total Count
1	kernel32.dll	2125	11	shlwapi.dll	514
2	user32.dll	1560	12	winspool.drv	357
3	advapi32.dll	1412	13	ws2_32.dll	283
4	gdi32.dll	1186	14	wininet.dll	267
5	shell32.dll	1115	15	shfolder.dll	221
6	oleaut32.dll	1085	16	mscoree.dll	195
7	ole32.dll	870	17	winmm.dll	192
8	comctl32.dll	831	18	psapi.dll	187
9	comdlg32.dll	703	19	crypt32.dll	143
10	version.dll	578	20	msvcrt.dll	128

Even though Table 20 shows that the import library “gdi32.dll” was in the top five most common import libraries, some curation was done in the library section and it was not selected for use as it is a graphical interface library, as the import functions from that library are unlikely to contain any special meaning towards malicious files. The import library “msvcrt.dll” was chosen for use in the preliminary experiments even though it was number twenty in the twenty most common import libraries. “msvcrt.dll” was chosen to be used as a comparison against the other import libraries which were in the top five of the most commonly used import library list.

4.4.1 Preliminary Experiments Stage One

The first stage of the preliminary experiments tested the five chosen import libraries “kernel32.dll”, “shell32.dll”, “user32.dll”, “advapi32.dll”, and “msvcrt.dll” at 5% injection of the total benign feature space. For each of the five import libraries, four experiments were performed which differed in how the import functions were select (manual or random) and the amount of import functions selected (five and 10). To reduce the amount of repeated information, the final results are summarised in two tables and the percentage of injection is only given for the “kernel32.dll” experiments.

4.4.1.1 Experiment 1.1 – Total Benign Feature Space Attack – Manual Selection

The first set of preliminary experiments in the first stage were developed to test the following two hypotheses:

H₁. A manual selection of features can be used for a successful adversarial poisoning attack.

H₃. No more than 5% of the benign training data feature space is required to be poisoned to reduce the general efficacy of the model.

The 5% of the benign training data feature space was calculated from the total amount of import functions in the benign training data (39,953,980), which was shown in Table 18.

For each import library selected, a selection of five and ten import libraries were chosen as the adversarial features which would be injected into the benign training data files. The import libraries were selected from the VirusShare dataset which supplied the Zero-day malware files and are detailed in Appendix A – . When examining the logs from the adversarial poisoning attack, it was found that the 5% injection threshold was never met, and that 100% of the benign training files which contained the target import library were injected with the adversarial features.

In experiment 1.1a, the five “kernel32.dll” import functions chosen for injection into each benign instance containing “kernel32.dll” were “WaitForSingleObject”, “SetFilePointer”, “WriteFile”, “ReadFile” and “GetModuleHandleA”. The total injection percentages among each of the training files is shown in Table 21, the total number of adversarial injections is 426,601, which is 4.372% of the target benign “kernel32.dll” feature space (9,756,993) and 1.068% of all the total benign feature space (39,953,980).

Table 21 - Kernel32.dll Five Functions Preliminary Poisoning Attack (Manual Selection)

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	19,100	1,248,297	46,761	3.745%
train_features_1.JSONI	26,354	1,740,959	75,924	4.361%
train_features_2.JSONI	29,605	1,852,481	87,451	4.720%
train_features_3.JSONI	26,892	1,562,435	74,754	4.784%
train_features_4.JSONI	28,480	2,104,726	71,005	3.373%
train_features_5.JSONI	22,939	1,248,095	70,706	5.665%
Total	153,370	9,756,993	426,601	4.372%

Experiment 1.1b was performed using “kernel32.dll” as the injection target for the poisoning attack and a manual selection of ten import functions. The ten “kernel32.dll” import functions chosen were “WaitForSingleObject”, “SetFilePointer”, “WriteFile”, “ReadFile”, “GetModuleHandleA”, “ExitProcess”, “GetProcAddress”, “GetLastError”, “LoadLibraryA” and “MultiByteToWideChar”. The total injection rate among each of the training files is shown in Table 22, with 723,518 functions being injected in total. The injection percentage across the benign import feature space (39,953,980 functions) and the benign and malicious import feature space (61,865,152 functions) was 1.810% and 1.170%.

Table 22 - Experiment 1.1b Injection Percentages

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	19,100	1,248,297	78,752	6.308%
train_features_1.JSONI	26,354	1,740,959	124,009	7.123%
train_features_2.JSONI	29,605	1,852,481	146,662	7.917%
train_features_3.JSONI	26,892	1,562,435	129,749	8.304%
train_features_4.JSONI	28,480	2,104,726	122,903	5.839%
train_features_5.JSONI	22,939	1,248,095	121,443	9.730%
Total	153,370	9,756,993	723,518	7.415%

From the results shown in

Table 23, the only import library which did not significantly increase the FNR of the ML model was the “msvcrt.dll” import library. The “msvcrt.dll” library was selected for a comparison as it contained the least number of instances in the VirusShare dataset, but as shown in Table 18, the msvcrt.dll library was in a similar amount of benign and malicious files as “shell32.dll” while containing almost five times the amount of import functions in both the benign and malicious training data. It is not clear why the “msvcrt.dll” poisoning attacks did not induce as many false negatives as the other attacks, but what is shown is that the results support the tested hypotheses H_1 , and H_3 .

Table 23 - Manual Import Selection - 5% Total Benign Feature Space

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
manual_kernel32_5	0.986	13.379	0.980	99.291	86.621
manual_kernel32_10	0.992	17.118	0.989	99.188	82.882
manual_user32_5	0.571	58.177	0.999	97.714	41.823
manual_user32_10	0.000	100.000	1.000	97.491	0.000
manual_shell32_5	0.965	34.630	0.992	98.959	65.370
manual_shell32_10	0.810	53.969	0.997	98.307	46.031
manual_advapi32_5	0.419	56.358	0.998	98.145	43.642
manual_advapi32_10	0.993	51.452	0.998	97.575	48.548
manual_msvcrt_5	0.997	4.114	0.802	99.757	95.886
manual_msvcrt_10	0.996	4.623	0.806	99.746	95.377

4.4.1.2 Experiment 1.2 - Total Benign Feature Space Attack – Random Selection

The second set of preliminary experiments in the first stage were developed to test the following two hypotheses:

H₂. A random selection of features can be used for a successful adversarial poisoning attack.

H₃. No more than 5% of the benign training data feature space is required to be poisoned to reduce the general efficacy of the model.

Experiment 1.2a was performed using “kernel32.dll” as the injection target for the poisoning attack. The five kernel32.dll import functions randomly chosen for injection were “VirtualFree”, “SetFilePointer”, “InterlockedDecrement”, “lstrlenA” and “IsDebuggerPresent”. The total injection rate among each of the training files is shown in Table 24 with 517,791 functions being injected in total. The injection percentage across the benign import feature space (39,953,980 functions) and the benign and malicious import feature space (61,865,152 functions) was 1.296% and 0.837%. In Table 24, the percentage of injected adversarial features is shown compared to the import library count and import library function count for “kernel32.dll” in the benign training files.

Table 24 - Experiment 2.1a Injection Percentages

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	19,100	1,248,297	61,311	4.911%
train_features_1.JSONI	26,354	1,740,959	92,312	5.302%
train_features_2.JSONI	29,605	1,852,481	104,149	5.622%
train_features_3.JSONI	26,892	1,562,435	89,945	5.756%
train_features_4.JSONI	28,480	2,104,726	85,185	4.047%
train_features_5.JSONI	22,939	1,248,095	84,889	6.801%
Total	153,370	9,756,993	517,791	5.306%

Experiment 2.1b was performed using “kernel32.dll” as the injection target for the poisoning attack. The ten “kernel32.dll” import functions randomly chosen for injection into each benign instance containing “kernel32.dll” were “VirtualFree”, “SetFilePointer”, “InterlockedDecrement”, “lstrlenA”, “IsDebuggerPresent”, “LocalFree”, “TerminateProcess”, “WideCharToMultiByte”, “GetCommandLineA” and “GetFileAttributesA”. The total injection rate among each of the training files is shown in Table 25, with 952,233 functions being injected in total. The injection percentage across the benign import feature space (39,953,980 functions) and the benign and malicious import feature space (61,865,152 functions) was 2.383% and 1.539%.

Table 25 - Experiment 2.1b Injection Percentages

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	19,100	1,248,297	112,493	9.011%
train_features_1.JSONI	26,354	1,740,959	164,798	9.465%
train_features_2.JSONI	29,605	1,852,481	187,832	10.139%
train_features_3.JSONI	26,892	1,562,435	169,003	4.416%
train_features_4.JSONI	28,480	2,104,726	162,061	7.699%
train_features_5.JSONI	22,939	1,248,095	156,046	12.502%
Total	153,370	9,756,993	952,233	9.759%

From the results shown in Table 26, the only import library which did not significantly increase the FNR of the ML model was the “msvcrt.dll” import library, this is the same result from the set of tests in experiment 1.1. The “msvcrt.dll” library was selected for a comparison as it contained the least number of instances in the VirusShare dataset, but as shown in Table 18, the msvcrt.dll library was in a similar amount of benign and malicious files as “shell32.dll” was while containing almost five times the amount of import functions in both the benign and malicious training data. It is not clear why the “msvcrt.dll” poisoning attacks did not induce as many false negatives as the other attacks, but what is shown that the results support the tested hypotheses H_1 , and H_3 .

Table 26 - Random Import Selection - 5% Total Benign Feature Space

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
random_kernel32_5	0.000	100.000	1.000	98.143	0.000
random_kernel32_10	0.000	100.000	1.000	97.867	0.000
random_user32_5	0.569	48.914	0.998	98.419	51.086
random_user32_10	0.264	60.938	0.999	98.034	39.062
random_shell32_5	0.944	21.954	0.991	99.153	78.046
random_shell32_10	0.944	21.954	0.991	99.153	78.046
random_advapi32_5	0.539	63.209	0.999	96.910	36.791
random_advapi32_10	0.353	65.769	0.999	97.225	34.231
random_msvcrt_5	0.998	2.876	0.689	99.843	97.124
random_msvcrt_10	0.998	4.346	0.786	99.763	95.654

4.4.2 Preliminary Experiments Stage Two

In the second stage of the preliminary experiments, the percentage of injection was changed from using the total benign feature space to the amount of the target benign files. As with the first stage of preliminary experiments, the same set of import functions from each test (manual and random) were used. The percentage of injection for each experiment started at 5% of the target benign files and continued in steps of 5% until 100%, with the final experiment being the same as the 5% benign feature space experiments.

4.4.2.1 Experiment 2.1a – Five Import Functions Manual

The second set of preliminary experiments were developed to test the following two hypotheses:

H₁. A manual selection of features can be used for a successful adversarial poisoning attack.

H₄. No more than 10% of the benign training data files are required to be poisoned to reduce the general efficacy of the model.

As the results from the previous preliminary experiments showed that trying to inject adversarial features into no more than 5% of the total benign feature space was the same as injecting adversarial features into 100% of the benign training data, the following set of experiments were performed, starting with 5% of the target benign files being injected with adversarial features and continuing in steps of 5% until 100%. The preliminary experiments were performed this way to see if there is an identifiable injection percentage which significantly impacts the target model.

The results from every test are detailed in Appendix A – , as there are too many tables to present in this section, the results from the “kernel32.dll” poisoning attack will be used as an example. The “kernel32.dll” import library was the most common library among all classes within the EMBER dataset. The five “kernel32.dll” import functions chosen for injection into each benign instance containing “kernel32.dll” were “WaitForSingleObject”, “SetFilePointer”, “WriteFile”, “ReadFile” and “GetModuleHandleA”.

Table 27 - Experiment 2.1a Results

Attack Name	FPR_1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
manual_kernel32_5_5	0.997	1.793	0.531	99.905	98.207
manual_kernel32_5_10	0.992	1.795	0.53 0	99.903	98.205
manual_kernel32_5_15	0.999	1.758	0.531	99.908	98.242
manual_kernel32_5_20	0.999	1.781	0.533	99.908	98.219
manual_kernel32_5_25	1.000	1.784	0.540	99.905	98.216
manual_kernel32_5_30	0.996	1.758	0.545	99.906	98.242
manual_kernel32_5_35	1.000	1.793	0.552	99.908	98.207
manual_kernel32_5_40	0.995	1.831	0.551	99.905	98.169
manual_kernel32_5_45	0.997	1.884	0.555	99.902	98.116
manual_kernel32_5_50	0.999	1.851	0.571	99.900	98.149
manual_kernel32_5_55	0.995	2.012	0.579	99.890	97.988
manual_kernel32_5_60	1.000	1.943	0.597	99.891	98.057
manual_kernel32_5_65	0.997	2.061	0.608	99.892	97.939
manual_kernel32_5_70	0.995	2.005	0.625	99.886	97.995
manual_kernel32_5_75	1.000	2.139	0.643	99.883	97.861
manual_kernel32_5_80	1.000	2.221	0.675	99.880	97.779
manual_kernel32_5_85	0.999	2.461	0.721	99.866	97.539
manual_kernel32_5_90	0.999	2.826	0.761	99.847	97.174
manual_kernel32_5_95	0.994	4.389	0.854	99.768	95.611
manual_kernel32_5_100	0.986	13.379	0.980	99.291	86.621

The results from the experiments in 2.1a, shown in Table 27, indicate that the efficacy of the model did degrade to a point which would render the model unusable, but only for the top injection rate of 100%. If using the threshold of 1% FPR, at 85% injection of the benign training files, the threshold increases from the clean model at 0.529 to 0.721. The 85% injection model still has a detection rate greater than 97.539%, but from a cursory glance it is obvious that something has gone awry during the training process. The results from the experiments support both H_1 and H_3 , as a manual selection of adversarial features was able to degrade the performance of the model with less than 5% of the total benign training data feature space being poisoned. The results from the experiments did not support H_4 , as the amount of benign training files which were poisoned for the successful attack exceeded 10% of the total benign training files.

4.4.2.2 Experiment 2.1b – Ten Import Functions Manual – kernel32.dll

Experiment 2.1b follows same process described for the previous “kernel32.dll” poisoning attack, with an addition of five import functions chosen for poisoning. The ten “kernel32.dll” import functions chosen for injection were “WaitForSingleObject”, “SetFilePointer”, “WriteFile”, “ReadFile”, “GetModuleHandleA”, “ExitProcess”, “GetProcAddress”, “GetLastError”,

“LoadLibraryA” and “MultiByteToWideChar”. The results from the experiments are shown in Table 28.

Table 28 - Experiment 2.1b Results

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
manual_kernel32_10_5	0.997	1.809	0.529	99.908	98.191
manual_kernel32_10_10	0.999	1.679	0.522	99.909	98.321
manual_kernel32_10_15	0.997	1.704	0.531	99.906	98.296
manual_kernel32_10_20	0.997	1.772	0.540	99.900	98.228
manual_kernel32_10_25	0.971	1.716	0.541	99.911	98.284
manual_kernel32_10_30	0.998	1.675	0.544	99.909	98.325
manual_kernel32_10_35	0.998	1.787	0.543	99.904	98.213
manual_kernel32_10_40	0.992	1.819	0.557	99.904	98.181
manual_kernel32_10_45	0.999	1.814	0.566	99.897	98.186
manual_kernel32_10_50	1.000	1.931	0.579	99.896	98.069
manual_kernel32_10_55	1.000	2.098	0.593	99.894	97.902
manual_kernel32_10_60	0.999	2.001	0.602	99.897	97.999
manual_kernel32_10_65	1.000	2.095	0.622	99.892	97.905
manual_kernel32_10_70	1.000	2.099	0.641	99.890	97.901
manual_kernel32_10_75	0.998	2.088	0.657	99.878	97.912
manual_kernel32_10_80	0.982	2.532	0.702	99.865	97.468
manual_kernel32_10_85	0.997	2.689	0.737	99.845	97.311
manual_kernel32_10_90	0.997	3.370	0.804	99.815	96.630
manual_kernel32_10_95	1.000	4.786	0.881	99.754	95.214
manual_kernel32_10_100	0.992	17.118	0.989	99.188	82.882

The results from the experiments in 2.1b show that the availability of the model did degrade to a point which would render the model unusable, but only for the top injection rate of 100%. If using the threshold of 1% FPR, at 85% injection of the benign training files, the threshold increases from the clean model at 0.529 to 0.737. The 85% injection model had an acceptable detection rate of just under 97.311%, but from a cursory glance it is obvious that something has gone awry during the training process. The results from the experiments support both H1 and H3, as a manual selection of adversarial features was able to degrade the performance of the model with less than 5% of the total benign training data feature space being poisoned. The results from the experiments did not support H4, as the amount of benign training files which were poisoned for the successful attack exceeded 10% of the total benign training files, as shown in Table 29.

Table 29 - Benign Training Files Injection Percentage - Manual Selection Attack

Import Library	Attack Success	Total Benign Training files Injected	Percent of Total Benign Training files
kernel32.dll	Yes	153,370	51.123%
shell32.dll	Yes	41,410	13.803%
user32.dll	Yes	84,533	28.178%
advapi32.dll	Yes	84,554	28.185%
msvcrt.dll	No	33,356	11.119%

4.4.2.3 Experiment 2.2a – Five Import Functions Random – kernel32.dll

The second set of preliminary experiments were developed to test the following two hypotheses:

H₂. A random selection of features can be used for a successful adversarial poisoning attack.

H₄. No more than 10% of the benign training data files are required to be poisoned to reduce the general efficacy of the model.

The second set of experiments using the step of 5% injection of the benign training file amount used the same set of adversarial import functions as the basis of the injection attack. The aim of the experiments was to see what level of degradation to the general efficacy of the model was achieved at the different percentages of adversarial injection. The results from each attack are shown in Table 30.

Table 30 - Experiment 2.1a Results

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
random_kernel32_5_5	1.000	1.820	0.528	99.907	98.180
random_kernel32_5_10	0.995	1.737	0.528	99.904	98.263
random_kernel32_5_15	0.994	1.815	0.538	99.905	98.185
random_kernel32_5_20	0.998	1.802	0.547	99.904	98.198
random_kernel32_5_25	0.999	1.869	0.556	99.905	98.131
random_kernel32_5_30	0.999	1.858	0.564	99.901	98.142
random_kernel32_5_35	0.997	1.952	0.586	99.899	98.048
random_kernel32_5_40	0.996	2.004	0.597	99.901	97.996
random_kernel32_5_45	0.999	1.969	0.602	99.894	98.031
random_kernel32_5_50	0.999	1.992	0.630	99.895	98.008
random_kernel32_5_55	0.999	2.041	0.648	99.892	97.959
random_kernel32_5_60	1.000	2.156	0.660	99.892	97.844
random_kernel32_5_65	0.996	2.248	0.689	99.880	97.752
random_kernel32_5_70	1.000	2.425	0.725	99.881	97.575
random_kernel32_5_75	0.996	2.386	0.751	99.880	97.614
random_kernel32_5_80	0.996	2.747	0.795	99.864	97.253
random_kernel32_5_85	0.997	2.877	0.836	99.859	97.123
random_kernel32_5_90	0.992	3.219	0.877	99.838	96.781
random_kernel32_5_95	0.989	5.344	0.941	99.753	94.656
random_kernel32_5_100	0.000	100.000	1.000	98.143	0.000

The results from the second set of experiments in 2.2a show that the availability of the model did degrade to a point which would render the model unusable, but only for the top injection rate of 100%. If using the threshold of 1% FPR, at 85% injection of the benign training files, the threshold increases from the clean model at 0.529 to 0.836. The 85% injection model still has a detection rate greater than 97.123%, but from a cursory glance it is obvious that something has gone awry during the training process.

The results from experiment support H_3 as the total amount of the benign training files feature space injected with the adversarial features did not surpass 5%. The results did not support H_4 , as the general efficacy of the model with ten percent of the benign training data being injected did not reduce the overall availability. The results do not mean that it is impossible to reduce the overall availability with ten percent of the benign training files being injected, but that the features chosen did not contain enough salient information or were too small an amount to reduce the overall availability of the model.

4.4.2.4 Experiment 2.2b – Ten Import Functions Random – kernel32.dll

Experiment 2.1b follows same process described for the previous “kernel32.dll” poisoning attack, with an addition of five import functions chosen for poisoning. The ten “kernel32.dll” import functions chosen for injection were “kernel32.dll” were “VirtualFree”, “SetFilePointer”, “InterlockedDecrement”, “lstrlenA”, “IsDebuggerPresent”, “LocalFree”, “TerminateProcess”, “WideCharToMultiByte”, “GetCommandLineA” and “GetFileAttributesA”. The results from the experiments are shown in Table 31.

Table 31 - Experiment 2.1b Results

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_kernel32_10_5	0.997	1.764	0.540	99.908	98.236
random_kernel32_10_10	1.000	1.737	0.534	99.906	98.263
random_kernel32_10_15	1.000	1.804	0.548	99.907	98.196
random_kernel32_10_20	1.000	1.827	0.551	99.904	98.173
random_kernel32_10_25	0.997	1.753	0.554	99.907	98.247
random_kernel32_10_30	0.995	1.847	0.564	99.901	98.153
random_kernel32_10_35	1.000	1.929	0.581	99.893	98.071
random_kernel32_10_40	0.998	2.010	0.599	99.895	97.990
random_kernel32_10_45	0.999	1.951	0.609	99.901	98.049
random_kernel32_10_50	0.999	2.077	0.635	99.883	97.923
random_kernel32_10_55	0.996	2.008	0.644	99.891	97.992
random_kernel32_10_60	0.994	1.971	0.657	99.895	98.029
random_kernel32_10_65	0.995	2.246	0.688	99.886	97.754
random_kernel32_10_70	1.000	2.372	0.733	99.874	97.628
random_kernel32_10_75	0.996	2.658	0.773	99.871	97.342
random_kernel32_10_80	0.997	2.858	0.806	99.863	97.142
random_kernel32_10_85	0.996	3.149	0.843	99.850	96.851
random_kernel32_10_90	0.989	3.513	0.892	99.821	96.487
random_kernel32_10_95	0.990	4.681	0.945	99.766	95.319
random_kernel32_10_100	0.000	100.000	1.000	97.867	0.000

The results from the first experiment in 1.1a (5% benign feature space) show that the availability of the model degraded in quality to a point which would render the model unusable. The results support hypothesis (H_1 , H_2) in that using a (manual, random) selection of features is adequate for performing a successful indiscriminate poisoning attack targeting the availability of a ML model.

The results from the second set of experiments in 1.1a show that the availability of the model did degrade to a point which would render the model unusable, but only for the top injection rate of 100%. If using the threshold of 1% FPR, at 85% injection of the benign training files, the threshold increases from the clean model at 0.529 to 0.843. The 85% injection model still has a detection rate

greater than 96.851%, but from a cursory glance it is obvious that something has gone awry during the training process.

The results from experiment support H_3 as the total amount of the benign training files feature space injected with the adversarial features did not surpass 5%. The results did not support H_4 , as the general efficacy of the model did not degrade with ten percent of the benign training data being injected with adversarial features.

Table 32 - Benign Training Files Injection Percentage - Random Selection Attack

Import Library	Attack Success	Total Benign Training files Injected	Percent of Total Benign Training files
kernel32.dll	Yes	153,370	51.123%
shell32.dll	Yes	41,410	13.803%
user32.dll	Yes	84,533	28.178%
advapi32.dll	Yes	84,554	28.185%
msvcrt.dll	No	33,356	11.119%

The percentages of benign training files being injected with adversarial features for the final attack (100% of the target benign files being injected) for each target import library are shown in Table 32. In every example the amount of benign training files injected for a successful attack exceeded 10% of the benign training files (300,000).

4.4.3 Preliminary Experiments Discussion

The following sections contains a summary of the results and how they relate to their corresponding research question / hypothesis. The format of the following two sections is the same, with the only difference being the approach for selecting the adversarial features (manual and random selection).

4.4.3.1 Manual Import Function Selection

For the first set of experiments, using a maximum of 5% of the total benign feature space was chosen as the threshold for the adversarial poisoning attack. No instances of the adversarial poisoning attack achieved an injection rate of 5%, and for each poisoning attack every benign training file which contained the targeted import library was injected with the adversarial functions. The results from the preliminary experiments using a manual selection of five and ten import functions is shown in Table 33.

Table 33 - General efficacy Attack - Feature Space Manual

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
advapi32_10_manual	0.993	51.452	0.998	97.575	48.548
kernel32_10_manual	0.992	17.118	0.989	99.188	82.882
user32_10_manual	0.000	100.000	1.000	97.491	0.000
msvcrt_10_manual	0.995	3.155	0.720	99.828	96.845
shell32_10_manual	0.810	53.969	0.997	98.307	46.031
advapi32_5_manual	0.419	56.358	0.998	98.145	43.642
kernel32_5_manual	0.986	13.379	0.980	99.291	86.621
user32_5_manual	0.571	58.177	0.999	97.714	41.823
msvcrt_5_manual	0.993	2.725	0.686	99.838	97.275
shell32_5_manual	0.965	34.630	0.992	98.959	65.370

The second set of experiments tested the general efficacy of the model after injecting adversarial features into percentages of the target benign feature space. The poisoning attacks were increasing in steps of 5%, starting at 5% and finishing at 100% of the benign files containing the targeted import library being injected with the adversarial functions.

The performance pattern of the second set of experiments held the same for each import library. The threshold and FNR slowly increased up until 100% of the benign training files containing the import library were injected. Once 100% injection was achieved, the FNR of the model jumped significantly from the rest of the experiments. The only import library which did not have a significant increase in FNR at 100% injection was “msvcrt.dll”. The highest FNR achieved in a manual selection “msvcrt.dll” attack was 4.623% at 100% of the benign training files containing the library being injected with the adversarial features.

The hypotheses being tested in manual import selection stages of the preliminary experiments were H_1 , H_3 , and H_4 which are stated below. The results from testing the hypotheses are shown in Table 34.

H_1 . A manual selection of features can be used for a successful adversarial poisoning attack.

H_3 . No more than 5% of the benign feature space is required to be poisoned to reduce the general efficacy of the model.

H_4 . No more than 10% of the training data is required to be poisoned to reduce the general efficacy of the model.

Table 34 - H1, H3, and H4 Results

Hypothesis	Related Experiments	Supported
H ₁	All	Yes
H ₃	All	Yes
H ₄	All	No

The only hypothesis which was not supported by the results from any of the experiments was H₄. None of the experiments were able to reduce the general efficacy of the model at an injection rate of 10% benign training files containing the targeted import library.

It was shown that it is possible to reduce the overall availability of the model from a manual selection of adversarial features which supports H₁, the overall amount of training files which needed to be injected when using the chosen sample set of adversarial features was significantly large. The high percentage of poisoning was likely due to the small number of features chosen for injection, as the imports section of Windows PE32 files in EMBER comprises the majority of the feature vectors used for training the model.

Additional experiments to identify what number of features are required to reduce the general efficacy of the model at a smaller injection percentage were not performed as one of the aims of the targeted Zero-day attack is to perform the attack in a black-box scenario. Identifying which features contribute the most to a model's classification would require having access to the target training dataset or a surrogate model trained using the same model architecture. As the targeted Zero-day attack uses Windows PE32 files as the basis of the attack, it is safe to assume that the imports section of the file will provide some level of contribution to the model's classification, as the feature engineering summarised in section 2.8 identified the imports section as a positive contributor for training ML malware detection models

H₃ was supported from the experiments as 5% of the benign feature space of the training data being injected with adversarial features was never achieved using the target manual selection of adversarial features. In the benign feature space experiments, it was found that every instance of benign training files which contained the target import library were injected with the adversarial features, the same as the 100% injection attack performed in the other set of preliminary experiments. The target import library "msvcrt.dll" was the only experiment to not reduce the overall availability of the model after injecting the adversarial features into every benign training file.

4.4.3.2 Random Import Function Selection

For the first set of experiments, a maximum of 5% of the total benign feature space was chosen as the threshold for the adversarial poisoning attack, while the adversarial features were randomly selected from the import library. No instances of the adversarial poisoning attacks achieved an injection rate of

5%, and for each poisoning attack every benign training file which contained the targeted import library was injected with the adversarial features. The results from the preliminary experiments using a random selection of five and ten import functions is shown in Table 35.

Table 35 - General efficacy Attack - Feature Space Random

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
advapi32_10_random	0.353	65.769	0.999	97.225	34.231
kernel32_10_random	0.000	100.000	1.000	97.867	0.000
user32_10_random	0.264	60.938	0.999	98.034	39.062
msvcrt_10_random	0.998	4.346	0.786	99.763	95.654
shell32_10_random	0.247	76.591	0.999	97.244	23.409
advapi32_5_random	0.539	63.209	0.999	96.910	36.791
kernel32_5_random	0.000	100.000	1.000	98.143	0.000
user32_5_random	0.569	48.914	0.998	98.419	51.086
msvcrt_5_random	0.998	2.876	0.689	99.843	97.124
shell32_5_random	0.608	63.109	0.998	97.892	36.891

The second set of experiments tested the general efficacy of the model after injecting adversarial features into percentages of the total benign training files containing the targeted import library. The poisoning attacks were increasing in steps of 5%, starting at 5% and finishing at 100% of the benign files containing the targeted import library being injected with the adversarial functions.

The performance pattern of the second set of experiments held the same for each import library. The threshold and FNR slowly increased up until 100% of the benign training files containing the import library were injected. Once 100% injection was achieved, the FNR of the model jumped significantly from the rest of the experiments. The only import library which did not have a significant increase in FNR at 100% injection was “msvcrt.dll”. The highest FNR achieved in a manual selection “msvcrt.dll” attack was 4.346% at 100% of the benign training files containing the library being injected with the adversarial features.

The hypotheses being tested in random import function selection stages of the preliminary experiments were H_2 , H_3 , and H_4 which are stated below. The results from the testing the hypotheses are shown in Table 36.

H_2 . A random selection of features can be used for a successful adversarial poisoning attack.

H_3 . No more than 5% of the benign feature space is required to be poisoned to reduce the general efficacy of the model.

H_4 . No more than 10% of the training data is required to be poisoned to reduce the general efficacy of the model.

Table 36 - H_2 , H_3 and, H_4 Experiment Relationship and Results

Hypothesis	Related Experiments	Supported
H_2	All	Yes
H_3	All	Yes
H_4	All	No

The only hypothesis which was not supported by the results from any of the experiments was H_4 . As it was with the preliminary experiments using a manual selection of features, none of the experiments were able to reduce the general efficacy of the model at an injection rate of 10% benign training files containing the targeted import library.

It was shown that it is possible to reduce the overall availability of the model from a random selection of adversarial features which supports H_2 , the overall amount of training files which needed to be injected when using the chosen sample set of adversarial features was significantly large. The results from the random selection of adversarial features were similar to the experiment results from the manual selection of adversarial features.

H_3 was supported from the experiments as 5% of the benign feature space of the training data being injected with adversarial features was never achieved using the target random selection of adversarial features. In the benign feature space experiments, it was found that every instance of benign training files which contained the target import library were injected with the adversarial features, the same as the 100% injection attack performed in the other set of preliminary experiments.

4.5 Main Experiments

As stated in Section 3.2.2, the fifth phase was the development of a targeted adversarial poisoning attack which allows for the targeted Zero-day malware file to bypass detection, while the general efficacy of the model is not significantly affected. The targeted adversarial poisoning attacks followed the process from the preliminary experiments, in that the imports section of benign files were injected with adversarial features and the selection of import libraries chosen for the targeted adversarial poisoning attacks is from the five libraries tested in the preliminary experiments.

The targeted adversarial poisoning attacks were performed on both the GBDT model architecture supplied with the EMBER dataset and on MLP model's which were trained using a subset of the data from the EMBER dataset. Two methods of injection were performed in the targeted adversarial poisoning attacks. The first method, which was called the individual poisoning attack, took a selection of randomly chosen import functions from the four chosen import libraries of the target Zero-day file ('shell32.dll', 'user32.dll', 'kernel32.dll' and 'advapi32.dll'), and injected them into the benign training files. The injection of adversarial features was performed as a separate process for each target

import library. This means that the poisoning process was performed four times, once for each of the chosen target import libraries, and that the selection of benign training files injected with the adversarial features was different for each poisoning process. Of course, it was possible for a benign training file to be injected with adversarial features from multiple import libraries if it happened to contain two or more of the target import libraries and was chosen for injection more than once.

The individual poisoning attack was developed to identify how well the targeted poisoning attack would perform if the adversarial features chosen for injection were spread out throughout the benign training data files, instead of being clustered together.

The second method, called the combined poisoning attack, was developed to test the inverse of the individual poisoning attack. In the combined poisoning attack, only benign training files which contained all four target import libraries were injected with the adversarial features. The adversarial features used in the individual poisoning attack were the same in the combined poisoning attack.

4.5.1 Individual and Combined Attack Tests

As stated in 3.2.2, the fifth phase was to test the targeted Zero-day adversarial attack. The premise behind the targeted Zero-day adversarial poisoning attack was influenced from how poisoning attacks had been performed on image recognition models. In the targeted image recognition poisoning attack performed by Gu, Liu, Dolan-Gavitt, and Garg (2019), a trigger mark was appended to images which were mislabelled by the attacker as their desired label. For example, a stop sign with a trigger mark would be incorrectly labelled as a speed limit sign, as shown in Figure 4.1. The ML model was trained with the mislabelled images in the training dataset which resulted in the model classifying the images as the target adversarial label instead of the correct label.



Figure 4.1 - Example of a BadNet Trigger Image Adapted from (Gu et al., 2019)

The classification model used in the targeted Zero-day poisoning attack is binary and not multiclass i.e., the model classifies Windows PE32 files as either benign or malicious, and not into a malware family or some other multiclass category. Other ML malware classification poisoning attacks have been performed by researchers by mislabelling malicious files as benign files during training, which causes the targeted malicious file to be misclassified as benign. This type of attack would likely be identified during pre-processing of the data through either dynamic or static analysis of the training samples.

The targeted Zero-day adversarial poisoning attack injects adversarial features into a subset of the benign training samples. The adversarial features are from the Zero-day malware file, with the aim of the attack being to misclassify the Zero-day file as benign if enough of the adversarial features are spread into the benign feature space to shift the classification from malicious to benign. The selected adversarial features are import functions from unique import libraries. Import functions were selected for injection as they would not break the structure of the file and by themselves would not appear to be malicious.

The poisoning attack was performed by randomly selecting import features from four unique libraries 'shell32.dll', 'user32.dll', 'kernel32.dll' and 'advapi32.dll' of a selection of Windows PE32 malware files obtained from VirusShare. The "msvcrt.dll" import library which was examined in the preliminary experiments was not chosen for the targeted adversarial poisoning attacks, as it was shown to have poor performance as a target poisoning library. The set of adversarial features were randomly selected using Python's in-built random module instead of manual, as the exploratory results from the preliminary experiments had shown that there was no significant difference between the two approaches, and the random approach is more suitable for a black-box attack scenario. The submission date of each malware is from 2019, to simulate a Zero-day malware file. Each malware file was originally classified as malicious by the clean EMBER model, with an EMBER score of over 0.9, with the threshold of the original clean EMBER model being 0.529.

Up to ten features from each library was randomly selected using the secure random python function for the poisoning attack. Not every library contained ten features, with the total and average number of functions from the four libraries shown in Table 37.

.

Table 37 - Import Functions - Targeted Attack

Library Name	Total Import Functions	Average Import Functions
Advapi32.dll	3352	6
Kernel32.dll	5162	9
User32.dll	4970	9
Shell32.dll	1669	3

4.5.1.1 Decision Tree Model Attacks

As stated in 3.2.2, the sixth phase was the testing of the targeted Zero-day adversarial attack on different ML algorithms. The two chosen ML algorithms were Gradient Boosted Decision Tree (GBDT), which was used in the original EMBER framework, and a Multi-Layer Perceptron (MLP) model.

Two types of adversarial poisoning attacks were performed on the GBDT EMBER model, they differed by how the benign files were selected for injection. In the first experiment, the import functions from each target import library were injected into a random selection of benign training files that contained the target import library. The injection of adversarial files was done separately, so the benign files which were injected with adversarial features from “advapi32.dll” may or may not also contain adversarial features from the other import libraries, this experiment was called the individual poisoning attack. The aim of the individual attack was not to generate a clear pattern for the adversarial example to belong to, but instead test if enough adversarial features injected sporadically throughout the test data, would be sufficient in inducing a targeted false negative at a significant rate.

The second experiment, which was called the combined poisoning attack, only injected the adversarial features into benign training files which contained all four target import libraries. The combined attack experiment reduced the amount of available target benign training files for poisoning but was designed to create a pattern within the benign class which when triggered by the Zero-day malware test file, would shift the files classification from benign to malicious.

4.5.1.2 Individual Poisoning Attack Approach (GBDT)

Before running the experiments on each of the 543 target Zero-day files obtained from VirusShare, a selection of 50 files was used to test both the individual and combined attacks at different percentages of the benign training file set. The experiments were performed in steps of 5%, starting at 5% and finishing at 25%. Unlike the exploratory preliminary experiments, where the percentage of injection went to 100% to test the different levels of poisoning, the targeted attacks are under more realistic constraints of a practical attack, and as such do not go to an excessive level of poisoning.

For the first set of experiments undertaking the individual poisoning attack approach, the first experiment at 5% injection rate resulted in no successful attacks, with 0.724 being the lowest score achieved, which was over the threshold (0.536) by nearly 20 points. The 25% injection experiment had 19 out of 50 successful attacks (38%), but the median threshold of the attacks was 0.664, which is 13.5 points higher than the original clean threshold of 0.529. Using a threshold boundary of 10% when retraining the model would show that something wrong, or something suspicious has occurred during training. The first ten results from the 5% poisoning attack are shown in Table 38 and the first ten results from the 25% injection attack are shown in Table 39.

Table 38 - 5% Poison Attack - First Ten Results

Virus_SHA256	Score	Threshold	Below Threshold	FPR	FNR
2bdc0256a49b00d768296fa96ab8cb69 5c55c0d888d9ca0f55e00db259d8e533	0.774	0.550	No	0.997	1.891
25267e28e177491b26c1926d5a3592e 60623fb870d6d984ff76db01ebfd6b08e	0.949	0.532	No	1.000	1.711
0a0ae6dbd8b19d0f4b3c9fd0d8e7d933 f3af14f0e28716d171f35b3e2e83c268	0.956	0.539	No	0.999	1.825
474dc611d96cab3d86cac6b07a0c0c4c 39a365a578e07c9a150b8a20c91f5ec4	0.964	0.530	No	0.997	1.793
4c7cdc1d141088c3eb9af66592c7da7f e3457a2e1862880d3d000805ffc32d6b	0.911	0.534	No	0.999	1.740
f61da4b8755d64742aa0517f7606dbd5 293efc7df498eb21d0493b52b85fa586	0.964	0.526	No	1.000	1.857
589cbfbca739ce9f99594b75356d5d3f 3920c37c3e2bda8681f6cbccc992c38e	0.827	0.551	No	1.000	1.855
9415462530a30caf7340aca71173a64c a5eaedafbb0d298835851f9cf42622cb	0.834	0.532	No	0.997	1.710
6bf29d8d7afc35f5ecf3ac1531158288 09487812392ef6bd82706c9c4422ef74	0.882	0.528	No	0.999	1.648
00c3f291834ed9f5bfeb52e9bd0ce78a d1fdc7bd106ab5e3024cf7e62700b034	0.864	0.541	No	0.997	1.871

Table 39 - 25% Poison Attack - First Ten Results

Virus_SHA256	Score	Threshold	Below Threshold	FPR	FNR
2bdc0256a49b00d768296fa96ab8cb69 5c55c0d888d9ca0f55e00db259d8e533	0.417	0.659	Yes	0.996	2.375
25267e28e177491b26c1926d5a3592e 60623fb870d6d984ff76db01ebfd6b08e	0.659	0.670	Yes	1.000	2.336
0a0ae6dbd8b19d0f4b3c9fd0d8e7d933 f3af14f0e28716d171f35b3e2e83c268	0.713	0.659	No	0.995	2.196
474dc611d96cab3d86cac6b07a0c0c4c 39a365a578e07c9a150b8a20c91f5ec4	0.818	0.672	No	0.999	2.489
4c7cdc1d141088c3eb9af66592c7da7f e3457a2e1862880d3d000805ffc32d6b	0.907	0.681	No	0.999	2.457
f61da4b8755d64742aa0517f7606dbd5 293efc7df498eb21d0493b52b85fa586	0.834	0.666	No	1.000	2.312
589cbfbca739ce9f99594b75356d5d3f 3920c37c3e2bda8681f6cbccc992c38e	0.352	0.660	Yes	0.997	2.332
9415462530a30caf7340aca71173a64c a5eaedafbb0d298835851f9cf42622cb	0.358	0.644	Yes	1.000	2.227
6bf29d8d7afc35f5ecf3ac1531158288 09487812392ef6bd82706c9c4422ef74	0.212	0.676	Yes	0.998	2.389
00c3f291834ed9f5bfeb52e9bd0ce78a d1fdc7bd106ab5e3024cf7e62700b034	0.293	0.659	Yes	0.999	2.345

The targeted Zero-day poisoning experiments are related to research question one and hypotheses five and six, which are restated below.

RQ1: Can adversarial attacks against machine learning based malware detectors increase the likelihood of unknown-unknown malware samples bypassing detection?

H₅. No more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.

As shown in Table 40 neither the 5% attack or the 10% attack had any significant number of successful targeted Zero-day poisoning attacks, the 5% test had no successful attacks and the 10% attack only had two. The individual attack approach was not expected to have any significant number of successful attacks from how the attack was performed but it was assumed that there would be more successful attacks than was shown. Neither of the hypotheses were supported by the results obtained from the first individual attack experiments.

Table 40 - Individual Library Poison Attack Results

Test	Total Successful Attacks	Average Score	Median Score	Average Threshold	Median Threshold
5%	0/50	0.901	0.918	0.539	0.538
10%	2/50	0.813	0.851	0.560	0.561
15%	9/50	0.748	0.832	0.586	0.589
20%	14/50	0.696	0.765	0.619	0.624
25%	19/50	0.645	0.713	0.657	0.664

4.5.1.3 Combined Poisoning Attack Approach (GBDT)

In the combined attack approach, the entire set of 543 Zero-day malware files were used in two sets of experiments. In the first set of experiments, the same benign training files were injected with the adversarial features for each targeted Zero-day poisoning attack. In the second set of experiments, a random seed was used for each Zero-day poisoning attack. The two sets of experiments were performed to see if there was a difference in the outcome of the attacks which could be related to the importance of the files injected with the adversarial features. In the combined attack approach, unlike the individual attack approach, only one percentage of the total benign training files was used for injection, which was 7.5% of the benign training files and 2.5% of the total training files.

There was a total of 40,551 benign files which contained each of the unique import libraries, as shown in Table 41.

Table 41 - Benign Train Data Containing All Target Import Libraries

Train Data	Benign Count	Four Library Count
Train_0	50,000	6174
Train_1	50,000	5144
Train_2	50,000	6046
Train_3	50,000	8306
Train_4	50,000	10299
Train_5	50,000	4582

The first experiment used the same seed for each adversarial injection and achieved a 60% success rate in misclassifying malicious files as benign, while keeping the threshold from exceeding 10% of the original clean model. The second experiment achieved a success rate of 58% while also keeping the threshold within 10% of the original clean model. There was a difference of 60 malware files which succeeded in bypassing classification in one test but not the other, with 38 malware files bypassing detection in the first experiment and 28 bypassing detection in the second. The overall results for both experiments are shown in Table 42. The results of the first ten tests in the first experiment are shown in Table 43 and the results of the first ten tests in the second experiment are shown in Table 44.

Table 42 - Combined Attack Results GBDT

Test	Total Successful Attacks	Average Score	Median Score	Average Threshold	Median Threshold
First Experiment	328/543	0.519	0.538	0.607	0.608
Second Experiment	312/543	0.531	0.551	0.606	0.606

Table 43 - Combined Attack GBDT - First Ten Results - Same Seed

Virus_SHA256	Score	Threshold	Below Threshold	FPR	FNR
2bdc0256a49b00d768296fa96ab8cb69 5c55c0d888d9ca0f55e00db259d8e533	0.568	0.619	Yes	0.998	2.158
25267e28e177491b26c1926d5a3592e 60623fb870d6d984ff76db01ebfd6b08e	0.677	0.592	No	1.000	2.044
0a0ae6dbd8b19d0f4b3c9fd0d8e7d933 f3af14f0e28716d171f35b3e2e83c268	0.452	0.614	Yes	1.000	2.135
474dc611d96cab3d86cac6b07a0c0c4c 39a365a578e07c9a150b8a20c91f5ec4	0.599	0.611	Yes	0.999	2.089
4c7cdc1d141088c3eb9af66592c7da7f e3457a2e1862880d3d000805ffc32d6b	0.618	0.605	No	0.999	2.259
f61da4b8755d64742aa0517f7606dbd5 293efc7df498eb21d0493b52b85fa586	0.599	0.606	Yes	0.992	2.074
589cbfbca739ce9f99594b75356d5d3f 3920c37c3e2bda8681f6cbccc992c38e	0.287	0.609	Yes	0.996	2.005
9415462530a30caf7340aca71173a64c a5eaedafbb0d298835851f9cf42622cb	0.149	0.608	Yes	0.993	1.972
6bf29d8d7afc35f5ecf3ac1531158288 09487812392ef6bd82706c9c4422ef74	0.265	0.612	Yes	0.996	2.038
00c3f291834ed9f5bfeb52e9bd0ce78a d1fdc7bd106ab5e3024cf7e62700b034	0.211	0.615	Yes	0.998	2.155

Table 44 - Combined Attack GBDT - First Ten Results - Random Seed

Virus_SHA256	Score	Threshold	Below Threshold	FPR	FNR
2bdc0256a49b00d768296fa96ab8cb69 5c55c0d888d9ca0f55e00db259d8e533	0.345	0.620	Yes	0.997	2.511
25267e28e177491b26c1926d5a3592e 60623fb870d6d984ff76db01ebfd6b08e	0.600	0.603	Yes	0.999	2.251
0a0ae6dbd8b19d0f4b3c9fd0d8e7d933 f3af14f0e28716d171f35b3e2e83c268	0.650	0.607	No	1.000	2.071
474dc611d96cab3d86cac6b07a0c0c4c 39a365a578e07c9a150b8a20c91f5ec4	0.596	0.599	Yes	0.999	2.020
4c7cdc1d141088c3eb9af66592c7da7f e3457a2e1862880d3d000805ffc32d6b	0.712	0.608	No	0.997	2.038
f61da4b8755d64742aa0517f7606dbd5 293efc7df498eb21d0493b52b85fa586	0.704	0.615	No	0.999	2.159
589cbfbca739ce9f99594b75356d5d3f 3920c37c3e2bda8681f6cbccc992c38e	0.266	0.608	Yes	1.000	2.099
9415462530a30caf7340aca71173a64c a5eaedafbb0d298835851f9cf42622cb	0.237	0.612	Yes	0.993	2.113
6bf29d8d7afc35f5ecf3ac1531158288 09487812392ef6bd82706c9c4422ef74	0.475	0.614	Yes	0.999	2.040
00c3f291834ed9f5bfeb52e9bd0ce78a d1fdc7bd106ab5e3024cf7e62700b034	0.251	0.615	Yes	0.999	2.198

The targeted Zero-day poisoning experiments are related to research question one and hypotheses five and six, which are restated below.

RQ1: Can adversarial attacks against machine learning based malware detectors increase the likelihood of unknown-unknown malware samples bypassing detection?

H₅. No more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.

The experiment results obtained from the combined attack approach were used using 7.5% of the benign training files which is related to H₅. From the results it was shown that performing the targeted Zero-day adversarial attack with less than 10% of the benign train data did produce successful results and over 60% of the Zero-day files which were previously classified correctly by the original clean EMBER2017 model were misclassified as malicious in their targeted poisoned ML model. The results from the experiments support the hypothesis H₅ that no more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.

4.5.1.4 Multi-Layer Perceptron Attack

As was stated in section 3.2.2, more than one ML algorithm was to be tested in the targeted Zero-day adversarial poisoning attacks. The first attack used the GBDT algorithm which was the default algorithm used in the EMBER framework. The additional ML algorithm chosen for use was an Artificial Neural Network (ANN). The type of ANN chosen is the basic multi-layer perceptron (MLP), which contains an input layer, one or more hidden layers, and a final output layer. The MLP was trained using the parameters shown below. The MLP network parameters were also adapted from a model which was also built by Endgame Inc, the same company responsible for EMBER. Using an MLP from Endgame instead of creating a new network architecture was done to assure uniformity between the experiments. The only changes made in the MLP parameters were to the hidden layers, an additional layer of ten neurons was included to provide a smaller layer for examination in the defence stage. The MLP model had the following parameters:

- Dense Model
- RELU Activation at hidden layers
- Sigmoid Activation at output layer
- Five hidden layers of [2048, 1024, 512, 256, 10] neurons
- Input_dropout=0.05
- Hidden_dropout=0.1
- Batch_size=128
- Epochs=100

The training data was comprised of 50,000 samples (25,000 benign and 25,000 malicious) which were extracted from the EMBER2017 dataset. The benign files were extracted to contain enough files to generate the adversarial examples containing the four chosen import libraries. These adversarial example files were extracted on a first-come basis and were not selected from a list of successful poisoning attacks from the GBDT experiments.

The dataset size of 50,000 was chosen to reduce the time it took to complete each experiment. The original dataset size was too big for the computer running the experiments to complete, and the GPU in the computer (AMD RX5700) was unable to support Keras. Performing the test on a selection of the original dataset also allowed for future tests to be performed using a different selection of training data. Out of the 543 Zero-day files, which were all detected as malicious by the GBDT model, two files were not detected by the MLP model, so they were not included in the experiment results.

The results from the clean model which are used as a baseline for the poisoning attacks are the following:

- ROC = 0.997
- Threshold = 0.725
- TPR = .963
- Accuracy @ Threshold = 0.976

The structure of the MLP poisoning experiments are as follows:

For the first set of experiments, a repeat of the individual poisoning attack at 25% of the benign training data was performed to provide some comparison data for the individual attack in the GBDT experiments. The 25% injection rate was chosen as it was the most successful percentage from the GBDT experiments. Even though 25% poisoning exceeds the 10% cut-off chosen in H₄, the experiments were still performed as they may produce valuable information.

The second, third and fourth set of experiments were to test the combined poisoning attack at 7.5%, 7.5% randomised and 15% of the benign training data. These values were chosen as the training dataset was curated with 15% of the files containing the four target import libraries. As the previous code used percentages of benign files which contained the target library, it was simple to change the code to select 50% and 100% of the training files which contained the target four import libraries. The third experiment was a repeat of the second experiment (7.5% poisoning), but in this instance the training data was randomly shuffled after poisoning using the shuffle function in python. The random shuffling was performed to remove clusters of the target benign files which existed in the curation of the training dataset. The second and third experiments were performed to compare the efficacy of the chosen adversarial features. If the second set of experiments, without the randomisation, had a significant increase in the number of successful attacks, then it is more than likely that the position of

the target training files influences the success of the poisoning attack more than the selection of features. If the third set of experiments has a similar number of successful attacks, then it is likely the selection of features played a more important role than if the target poisoned data clustered together.

When performing the second, third and fourth experiments, a line of code from the individual poisoning experiments was accidentally left in place which altered how the poisoning attack was performed. For training datasets train_1.jsonl, train_2.jsonl, train_3.jsonl, train_4.jsonl, and train_5.jsonl, the combined poisoning attack process was performed successfully. For the train_0.jsonl dataset, the combined poisoning process was followed when poisoning 'advapi32.dll', 'shell32.dll', and 'kernel32.dll', but for 'user32.dll' the code to generate the list of target files in the individual attack was accidentally left in place. As a result, the combined attack process succeeded for 'advapi32.dll', 'shell32.dll', and 'kernel32.dll', but for 'user32.dll' a new list was generated which included every benign file in train_0.jsonl that contained 'user32.dll' (1,327 files), and then a percentage of that total (which is greater than the total of benign training files that contained the target four import libraries) was poisoned instead, which lead to approximately 8.6% of the benign training files benign poisoned for each attack. As experiment two, three and four contained both the combined and individual approach, they will be referred to as the hybrid attack.

An additional experiment was undertaken which performed the combined poisoning attack as it was originally intended. In this experiment 10% of the benign training data was selected to be poisoned, which reflects (hypothesis number). This experiment was also performed by randomly shuffling the training data, unlike the third set of experiments, which poisoned the same benign training files as the second set of experiments, then randomly shuffled afterwards to compare the results. The final set of experiments randomly shuffled the training data before poisoning the benign training data.

4.5.1.5 Individual Poisoning Attack (MLP)

The first set of experiments was performed as an individual injection attack following the same process in section 4.5.1.1, but only at an injection rate of 25%. As the dataset was generated with at least 15% of the benign files containing a combination of the four chosen import libraries (shell32.dll, user32.dll, advapi32.dll and kernel32.dll), the poisoned benign files should cluster more significantly compared to the GBDT individual poisoning attack.

The first ten results are shown in Table 45. In comparison to the results from the GBDT attack shown in Table 40, the threshold from the MLP attacks is much higher, which is understandable as the clean GBDT model had a threshold of 0.529 and the clean MLP model had a threshold value of 0.725. In the first test, the poisoned training data generated were not randomised before training the ML model. The attack had 211/541 Zero-day files misclassified as benign, a success rate of 39.00% while not negatively influencing the general efficacy of the model.

Table 45 - MLP Combined Attack Approach - First Ten Results - Same Seed

Virus_SHA256	Score	Threshold	Below Threshold	FPR	FNR
2bdc0256a49b00d768296fa96ab8cb69 5c55c0d888d9ca0f55e00db259d8e533	0.271	0.861	Yes	1.000	4.36
25267e28e177491b26c1926d5a3592e6 0623fb870d6d984ff76db01ebfd6b08e	0.995	0.883	No	1.000	4.480
0a0ae6dbd8b19d0f4b3c9fd0d8e7d933f 3af14f0e28716d171f35b3e2e83c268	0.996	0.79 0	No	1.000	3.200
474dc611d96cab3d86cac6b07a0c0c4c 39a365a578e07c9a150b8a20c91f5ec4	0.994	0.861	No	1.000	3.880
4c7cdc1d141088c3eb9af66592c7da7fe 3457a2e1862880d3d000805ffc32d6b	0.993	0.873	No	1.000	3.880
f61da4b8755d64742aa0517f7606dbd5 293efc7df498eb21d0493b52b85fa586	0.995	0.850	No	1.000	4.200
589cbfbca739ce9f99594b75356d5d3f3 920c37c3e2bda8681f6cbccc992c38e	0.127	0.865	Yes	1.000	4.760
9415462530a30caf7340aca71173a64c a5eaedafbb0d298835851f9cf42622cb	0.173	0.827	Yes	1.000	3.320
6bf29d8d7afc35f5ecf3ac15311582880 9487812392ef6bd82706c9c4422ef74	0.639	0.901	Yes	0.960	3.840
00c3f291834ed9f5bfeb52e9bd0ce78ad 1fdc7bd106ab5e3024cf7e62700b034	0.217	0.862	Yes	1.000	4.360

4.5.1.6 Hybrid Poisoning Attacks (MLP)

There was a total of three sets of hybrid experiments performed. Each set of experiments tested 541 Zero-day poisoning attacks by using a combination of the combined and individual poisoning process. The hybrid attacks were initially designed to follow the combined poisoning approach, but a typo left in the code randomised the poisoning of the adversarial ‘user32.dll’ features in the train_0.jsonl dataset. In the first set of hybrid attack experiments, approximately 8.6% of the benign training data was injected with adversarial features for each experiment. The Training data at this stage had not

been randomised, so the target benign files for the combined attack clustered towards the beginning for each training dataset (train_0.jsonl – train_5.jsonl). The first set of hybrid attack experiments achieved a success rate of 37.33%, with 202 out of 541 successful Zero-day poisoning attacks.

In the second set of hybrid attack experiments, the same format from the previous set of experiments was followed, with the difference being that the training data had been randomly shuffled to reduce the number of clusters which existed due to the first come format used to generate the original training dataset. Each attack injected the same adversarial features into the same benign training file as the previous attack, once the entire training dataset was poisoned, it was randomly shuffled, and the model was trained. The attacks were performed that way to provide insight into the efficacy of the poisoning attack method. If the attacks achieved a similar success rate as the previous attack, it would likely mean that the percentage of files and chosen features were responsible for the attacks success. If on the other hand the previous attack contained more successful attacks while using the same adversarial features and which were injected into the same benign training files, then the position of the poisoned files influenced the success of the attacks, more so than the adversarial features. The second set of hybrid poisoning attack experiments achieved a success rate of 33.27%, with 184 out of 541 successful Zero-day poisoning attacks.

The third set of hybrid attack experiments, approximately 17.2% of the benign training data was injected with adversarial features. The aim of the second set of hybrid experiments was originally to see what would happen if 100% of the benign training files which contained the four target import libraries (15% of the benign training data) was injected with the adversarial features. As the typo existed in the set of experiments, there was an increase of benign training files which contained ‘user32.dll’ that were injected with adversarial features. The third set of hybrid attack experiments achieved a success rate of 80.59%, with 436 out of 541 successful Zero-day poisoning attacks. Unlike the previous sets of experiments, the success of the second set of experiments was due to the significant increase in the threshold, which is shown in Table 46.

4.5.1.7 Combined Poisoning Attack (MLP)

The combined poisoning attack injected adversarial features into 10% of the benign training data. Each benign training file which was poisoned contained the four import libraries ‘kernel32.dll’, ‘user32.dll’, ‘shell32.dll’, and ‘advapi32.dll’. A selection of import functions from the four import libraries were randomly selected from each target Zero-day file and injected into the target benign files. The training data was randomly shuffled before being poisoned to prevent the target benign files from clustering together. The combined attack experiments achieved a success rate of 24.95%, with 135 out of 541 successful attacks. The combined attack experiments had fewer successful attacks compared to both hybrid attacks which injected adversarial features into a smaller number of benign

training files. It is likely that the increased number of training files poisoned with adversarial features from ‘user32.dll’ influenced the success of the hybrid poisoning attacks.

Table 46 - Targeted Adversarial Poisoning Attack Results - MLP

Test Name	Total Successful Attacks	Average Score	Median Score	Average Threshold	Median Threshold	Average FNR
Individual Attack (23%)	211/541	0.683	0.986	0.856	0.858	4.040
Hybrid Attack (8.6%)	202/541	0.751	0.998	0.834	0.835	3.994
Hybrid Attack Randomised (8.6%)	184/541	0.823	0.997	0.811	0.813	3.358
Hybrid Attack (17%)	436/541	0.866	0.963	0.984	0.988	18.569
Combined Attack (10%)	135/541	0.914	0.993	0.837	0.836	4.049

When comparing the results from the randomised hybrid and combined set of experiments, it is clear that the hybrid attack experiments reduced the score of the Zero-day files significantly more so than the combined attack experiments. The difference in results from the successful Zero-day attacks is shown in Table 47, where the hybrid attacks on average are 0.317 points below the threshold compared to the combined attacks 0.090 points below. These results indicate that the extra number of adversarial features in the ‘user32.dll’ import library significantly influence the impact of the poisoning attack.

Table 47 - Randomised Hybrid and Combined Results Comparison

Experiment	Average Score Difference from Threshold	Min Difference from Threshold	Max Difference from Threshold	Average Successful Adversarial Score
Combined	0.090	0.00007	0.834	0.746
Hybrid	0.317	0.00279	0.835	0.449

The targeted Zero-day poisoning experiments are related to research question one and hypotheses five and six, which are restated below.

RQ1: Can adversarial attacks against machine learning based malware detectors increase the likelihood of unknown-unknown malware samples bypassing detection?

H₅. No more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.

The results from the targeted Zero-day poisoning experiments show that it is possible to increase the likelihood of Zero-day malware samples bypassing detection from ML trained malware detection models. In both the GBDT and MLP trained models, successful attempts to bypass classification through poisoning of the benign training set was achieved. The experiments did require up to ten percent of the benign training data samples to be infected with adversarial features, which supports H₅. The process of poisoning the model was performed in a black-box scenario, which reduced the ability to identify which features would contribute the most to the model's classification and to target those features specifically. If feature importance was to be performed before the poisoning attack, it is likely that the percentage of poisoned data required for a successful attack would be reduced. In the feature analysis paper by Saxe and Berlin (2015) it was found that the imports section when used by itself to identify patterns and correctly classify malicious files, performed the worst compared to the four other areas that were tested. In the experiments the imports section was chosen as it compromised the majority of the input feature vectors and in a practical sense, adding an import function to a binary PE32 file is a simple way to perform an adversarial poisoning attack. In future work other areas of the binary PE32 file would be tested to identify which area is best suited for manipulation to generate a successful targeted adversarial poisoning attack.

4.6 Defensive Strategies

As stated in Section 3.2.2, the seventh phase was the development and testing of defensive strategies to mitigate the Zero-day malware poisoning attack. Three different defences were explored to identify if it was possible to prevent the Zero-day malware poisoning attack at test time. A variety of defences

have been developed which identify poisoning attacks at the training stage, which was covered in 2.7.2, but defences at the testing stage has been ignored. This is likely due to the fact that it is preferable for a supposedly clean ML model to go live, than a potentially poisoned model. As it is not possible to be absolutely certain that a trained ML model is clean, even after applying a defensive strategy and removing identified poisoned data, the defences introduced in the following sections explore the possibility of identifying targeted poisoning attacks as they occur at test time, while keeping the model live.

4.6.1 Heatmaps

As stated in section 3.2.2, the seventh phase was the development of an adversarial defensive strategy to defend against the targeted Zero-day adversarial attack. The first approach for defending against the MLP poisoning attack was to analyse the activation weights in a heat map at the different layers, in the same way image recognition models are analysed to show on which section of an image the model is concentrating. The idea behind the heat map was to see if it is possible to distinguish a targeted Zero-day file from an actual benign file by how the weights are activated. An assumption was made that the Zero-day file, which had not been modified in any way, should contain a significant amount of activation weights which would lead the classifier correctly towards the malicious class, but are overwhelmed by a smaller, but stronger group of activation weights which belong to the injected adversarial features.

The tool keract is used to generate heatmaps of MLP activation weights for image recognition models which was selected to produce the heatmap of the Zero-day activation weights. As shown in Figure 4.2 1D representations of activation weights in a heatmap do not provide adequate information compared to a 2D image.

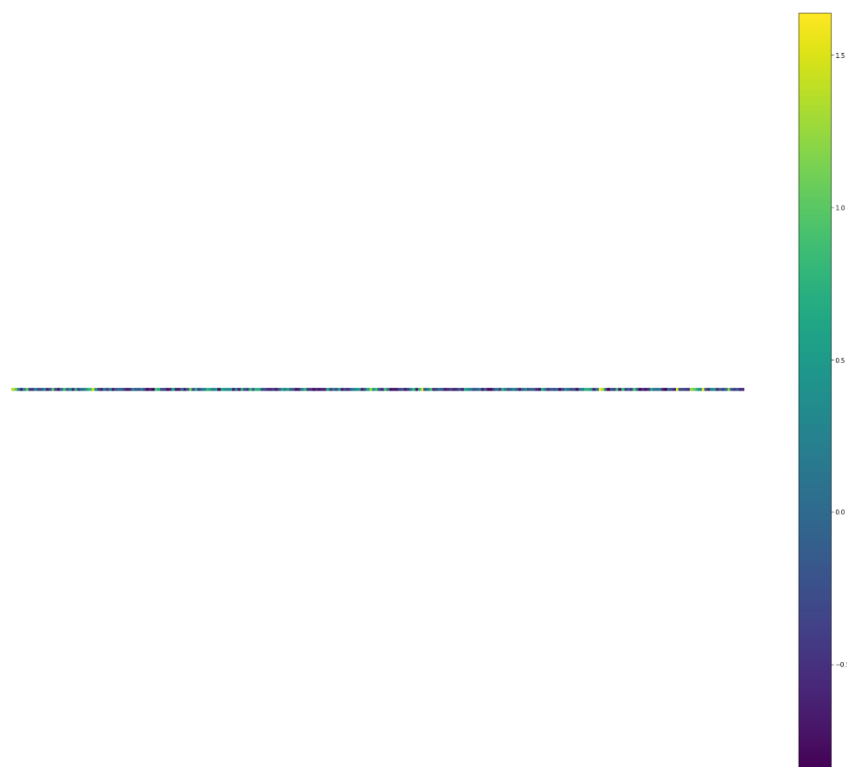


Figure 4.2 – Single Dimension Heatmap Representation of Activation Weights

4.6.2 Feature Importance

Another defensive approach was to identify the importance of features in determining the classification of data at test time. The approach was to use some form of permutation importance to identify which features contribute the most to the decision making. Permutation importance works by replacing or removing a feature or set of features in the test file and retesting the file for each set of changes. If the loss increases without the feature(s) being present, it is assumed that they positively contribute to the classification.

The defensive approach of feature importance was examined to see if it is possible to identify features within an adversarial example that belong to the trigger set and contribute to the misclassification of the file. That is to say, if removing certain features from a file changes the classification from benign to malicious, would that indicate that a targeted poisoning attack has been successfully performed? This is under the assumption that when running feature permutation on a clean model, removing features would only show the contribution to their classification, and that the class would not change from removing features. The class would only change in the targeted poisoning attack samples due to the trigger features being removed. As this defence would require removing each feature or set of features from a file and analysing the file for each permutation, it is not a defence which could be

reasonable performed at test time, and is best left identifying adversarial data within the training dataset.

4.6.3 Activation Clustering

4.6.3.1 At training

B. Chen et al. (2018) developed a defence for adversarial poisoning attacks which works by examining clusters of the activation weights from the last hidden layer. The defensive strategy assumes that the adversarial example's weights will cluster towards both the target adversarial class and the original source class, which is being attacked, compared to clean examples which will cluster towards their true target class. The defence was built for image recognition models, where a trigger mask is overlaid on target images to shift the classification towards the attackers desired class e.g., a stop sign being misclassified as a speed limit sign. In the case of the Zero-day poisoning attack, the adversarial features injected into the benign test data are features which can be found in both malicious and benign files. The cluster of the training data activation weights would likely show a strong benign cluster.

The poisoning defence has been integrated into the IBM Adversarial Robustness Toolbox. The defence was attempted on the MLP poisoned EMBER dataset but did not work due to a bounds limit on the dataset. The poisoning defence has a set limit of 5,000 and the MLP EMBER dataset contains 50,000 samples.

It is assumed that the activation clustering defence would not succeed in detecting the adversarial examples in the benign training dataset, as the adversarial examples are not mislabelled data e.g. a stop sign labelled as a speed limit sign, but genuine benign files injected with additional import functions. The injected import functions represent a small portion of the benign file's feature space, and do not alter the functionality of the benign file into a malicious file. A modified version of the activation clustering defence was performed by Severi et al. (2020) and it was found that the defence was unable to detect the watermark attack when using their combined approach (detailed in 2.7.1.8), as the adversarial features were too well camouflaged and became indistinguishable from genuine features.

An extension of the defence to be applied at test has been proposed to defend against targeted adversarial poisoning attacks for Zero-day malware files. Using the same premise as the activation clustering defence, the activation weight values from the neurons in the last hidden layer are to be examined at test time for anomalous behaviour. If the activation weight values of files classified as benign differ significantly from the average activation weight values of the benign class, and lean more towards the malicious class for a majority of the neurons, then the files are flagged as suspicious and to be quarantined for further examination.

4.6.3.2 At test time

A proposed idea is to first examine the average and normal distribution of each individual weight from the last hidden layer to find the norm to which a benign file should belong, and then to compare each of the individual weights for outliers and if they are significant enough to warrant further examination of the file (e.g., quarantine for further dynamic analysis). The assumption of why outliers in the activation weights would exist is due to how the targeted poisoning attack was performed. In the attack only the benign training data was injected with the adversarial features obtained from the Zero-day file, the Zero-day file has not been modified in any way, and originally was correctly classified as malicious. Since the Zero-day file has not been modified, the majority of the features present within the file should contribute towards a correct classification of malicious, with the trigger features overwhelming the correct classification due to their significant presence in the benign training data.

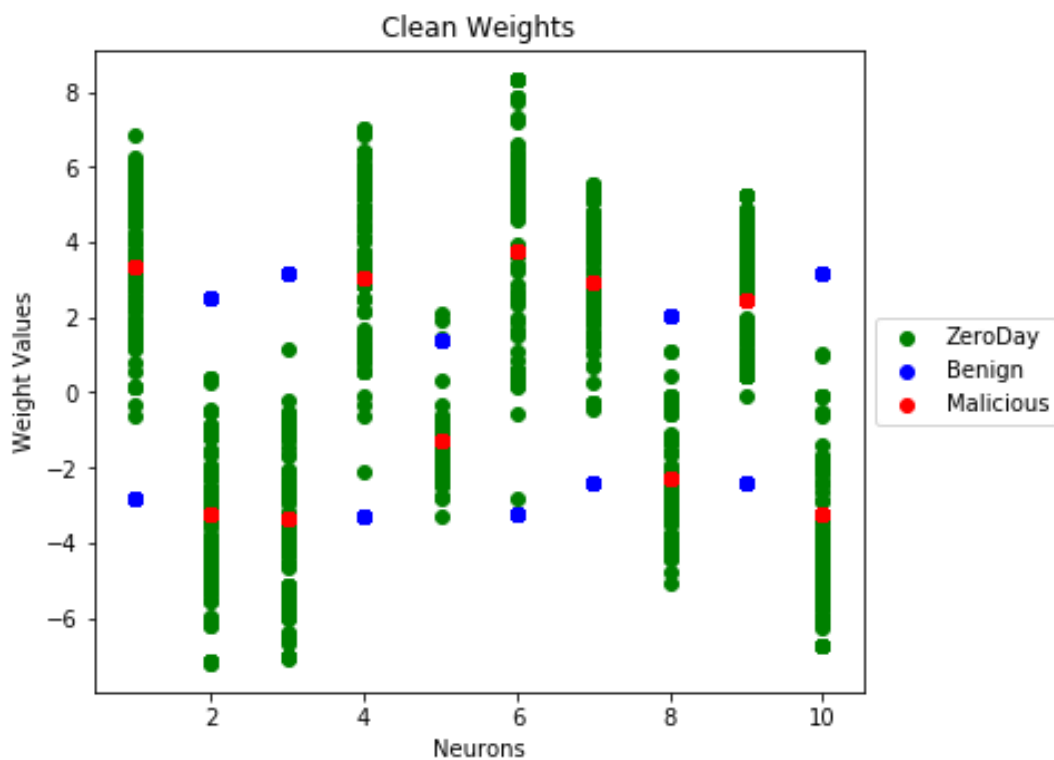


Figure 4.3 - Zero-day Activation Weights

Figure 4.3 illustrates the activation weights of the Zero-day files from the last hidden layer from the clean model overlayed on the average activation weight values from the training data. As the figure shows, the weights cluster towards the average malicious activation weight values.

Figure 4.4 and Figure 4.5 illustrate the activation weights of two Zero-day files which were accurately classified as malicious after 8.6% of the benign training files were injected with adversarial features from the Zero-day files.

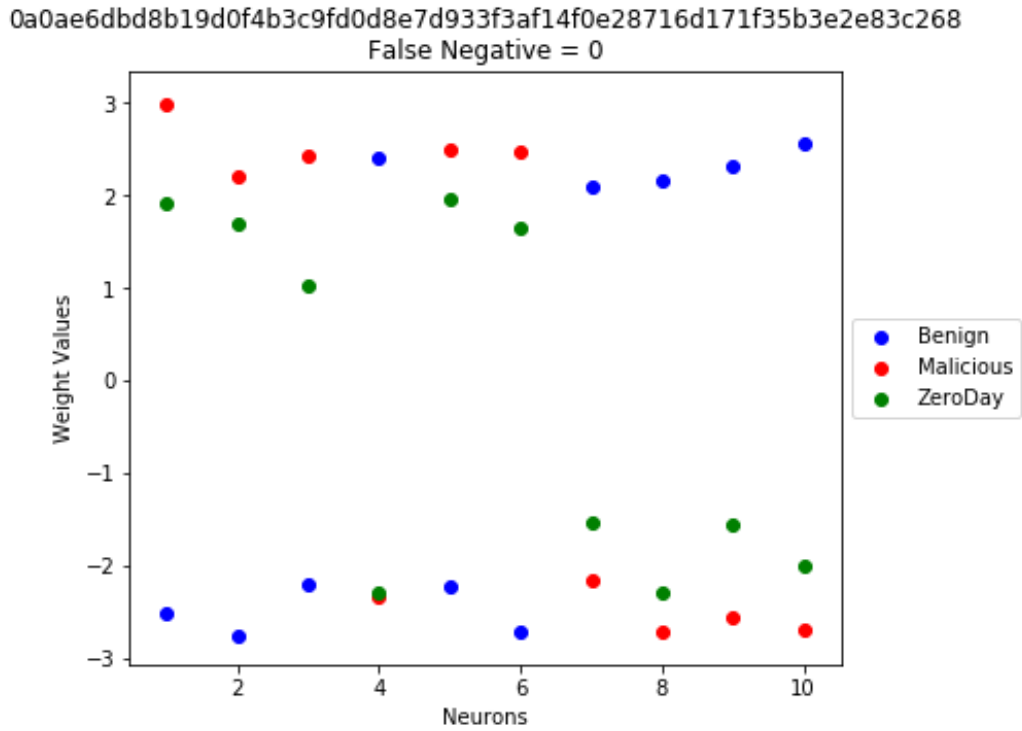


Figure 4.4 - Activation Weights of Zero-day Correct Classification - Example 1

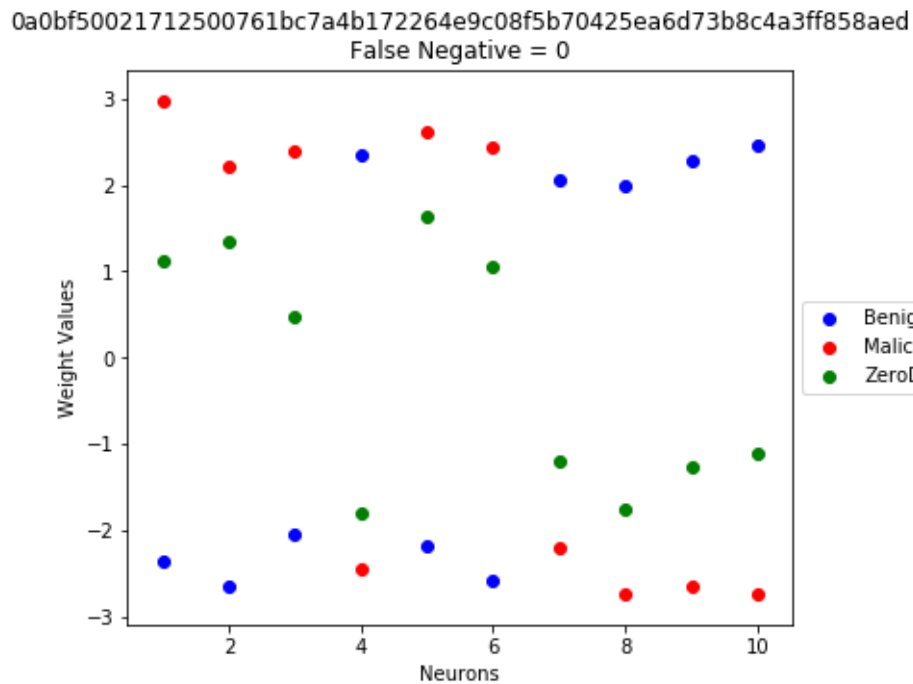


Figure 4.5 - Activation Weights of Zero-day Correct Classification - Example 2

Figure 4.6 and Figure 4.7 illustrate the activation weights of two Zero-day files which were misclassified as malicious after 8.6% of the benign training files were injected with adversarial

features from the Zero-day files. The Zero-day weights appear to centre between the benign and malicious activation weight averages.

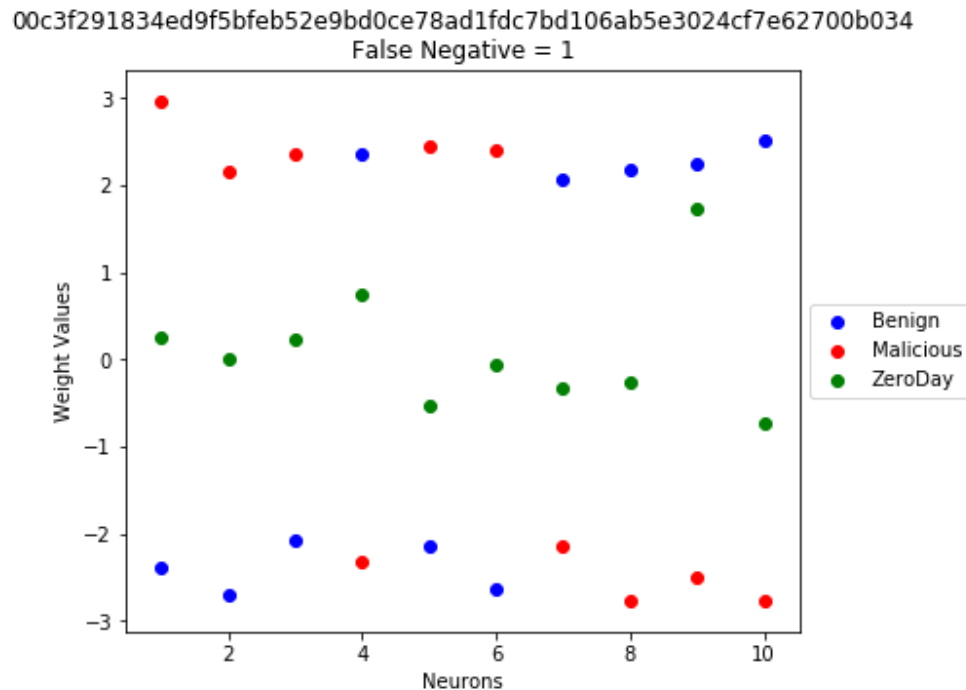


Figure 4.6 - Activation Weights of Zero-day Successful Mis-Classification - Example 1

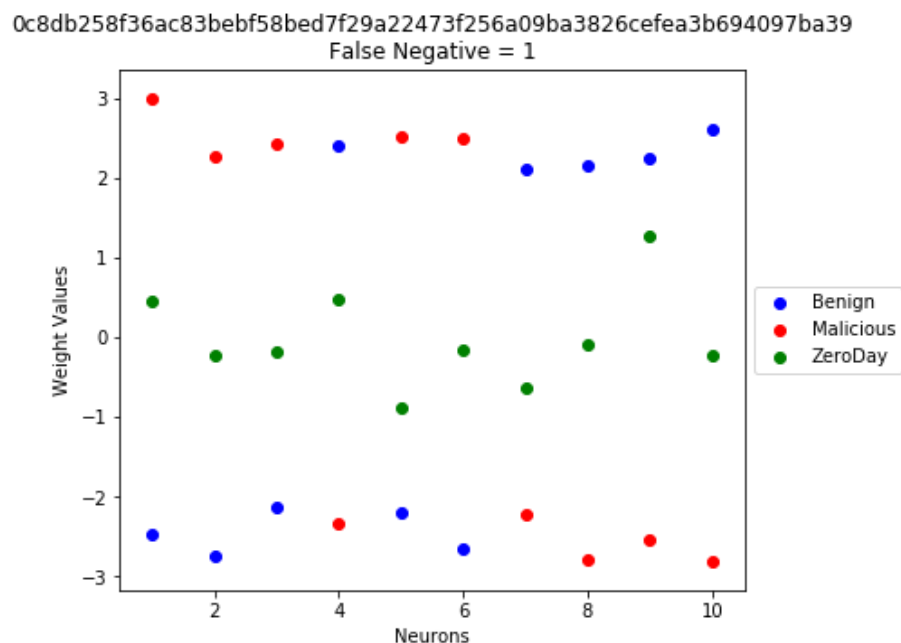


Figure 4.7 - Activation Weights of Zero-day Successful Mis-Classification - Example 2

After visually examining the activation weights from the successful Zero-day attacks, the weights were analysed to determine if they were within one standard deviation from the mean of the benign

activation weights. From the 184 successful attacks from experiment set three (hybrid poisoning attack randomised), 180 (97.82%) of the Zero-day files had eight or more activations weights outside of one standard deviation from their corresponding benign neuron activation weight value. Using the first successful attack as an example, the benign activation weights from the training dataset were examined and a total of 1,553 from 25,000 (6.21%), which is near the percentage of adversarial examples in the benign training data, were outside the specified range. To determine if the activation weights were outside the range in the benign training data was due to the poisoning attack, the activation weights of the benign training files from the clean model were examined, and 1,817 files (7.26%) had eight or more activation weights outside of the range, which is larger than the poison example.

After using the benign activation weights of the 25,000 files used to train the model, additional experiments were performed using the 100,000 benign test data files from the original EMBER2017 dataset. When calculating the averages and standard deviation of the activation weights from the 100,000 benign test files for each poisoned model, it was found that the averages of the activation weights were similar across each experiment, but the standard deviation was significantly different. From using a larger dataset, a greater amount of variance was introduced which included a significant number of outliers which prevented the previously successful defensive technique from working.

To remove the issue of the outliers in the dataset rendering the defensive technique unusable, another method was examined to identify if it was possible to detect suspicious benign classifications from examining the activation weights at the last hidden layer during test time. The method which was chosen for use is known as Mean Absolute Deviation (MAD). The MAD was chosen as it is another technique for identifying anomalies in data without the deviation being influenced by the present outliers in the dataset to such a significant degree. The MAD is the average of the distance between each value and the mean, the formula for calculating the MAD is shown below:

$$\frac{\sum |x - \bar{x}|}{n}$$

An example of the calculated MAD and standard deviations for the 100,000 activation weights of benign test files from a sample of successful targeted Zero-day poisoning attacks is shown in Table 48.

Table 48 - MAD Neuron Defence Examples

Virus	2bdc0256a49b00d768296fa96ab8cb695c55c0d888d9ca0f55e00db259d8e533									
Average	-2.54389	-2.69337	-2.25676	2.211975	-2.19205	-2.7541	2.063792	2.002713	2.146433	2.556544
Stdev	2.364112	3.772115	2.929054	2.082663	1.03714	4.643837	6.225784	1.360692	1.749687	3.87158
MAD	0.799923	0.805531	0.80593	0.802687	0.787503	0.805705	0.812789	0.808571	0.808331	0.814831
Virus	589cbfbca739ce9f99594b75356d5d3f3920c37c3e2bda8681f6cbccc992c38e									
Average	-2.37618	-2.60961	-2.03221	2.224023	-2.02518	-2.67347	2.008859	1.888502	2.038259	2.526909
Stdev	3.005188	3.813322	2.956617	2.318878	1.067834	4.091844	6.345914	1.208832	1.727433	3.234396
MAD	0.761926	0.776917	0.788561	0.781026	0.767752	0.786423	0.79369	0.789182	0.788709	0.797184
Virus	9415462530a30caf7340aca71173a64ca5eaedafbb0d298835851f9cf42622cb									
Average	-2.50684	-2.70737	-2.25143	2.207822	-2.21875	-2.75663	2.09072	1.994289	2.150688	2.562616
Stdev	2.199961	4.158785	2.197226	1.837669	1.046023	4.373703	6.720578	1.173849	2.204602	3.847392
MAD	0.78988	0.80404	0.802507	0.801294	0.787647	0.805115	0.812704	0.80945	0.810126	0.816471
Virus	00c3f291834ed9f5bfeb52e9bd0ce78ad1fdc7bd106ab5e3024cf7e62700b034									
Average	-2.42045	-2.64427	-2.07855	2.260539	-2.06865	-2.72758	2.003464	1.92851	2.085095	2.582588
Stdev	2.62585	4.061219	2.479173	2.221951	1.056772	4.717379	6.073107	1.706256	1.684715	3.196454
MAD	0.763034	0.779633	0.793342	0.785683	0.772798	0.793936	0.80351	0.79448	0.795314	0.805225
Virus	96ff3db9c22d16065a8f8e842d896508ae8bd7622e94330215912befbd7adf66									
Average	-2.5851	-2.71763	-2.2806	2.250088	-2.22894	-2.77706	2.041697	1.992896	2.12833	2.564581
Stdev	1.791967	4.034525	2.387503	1.17517	1.129382	4.813237	6.994591	1.137338	2.683498	4.163978
MAD	0.792863	0.801257	0.796823	0.799749	0.784861	0.801088	0.817177	0.816006	0.81796	0.823385

For each of the four sets of experiments performed on the MLP model, two defensive strategies were tested using the MAD. The defensive strategies followed the same formula as the original defence i.e., examining the activation weights from the last hidden layer of the target Zero-day file, but the MAD defence also tested to see if the activation weights of the neurons were outside of two MADs, instead of the single standard deviation which was originally used. The two MADs were also used as the general rule for identifying outliers is if they are two standard deviations away from the mean.

From examining the activation weights of the Zero-day files from Experiment 1 - Individual Attack (25%), which had 211 successful attacks out of 541 attempts, it was found that 105 of the successful Zero-day poisoning attacks could be identified by examining six neurons outside of both one and two MADs from the benign activation weights average. The results are shown in Table 49.

Table 49 - Individual Attack (25%) Defence Results

Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Benign Test Average FP	Benign Test FPR
120/211	56.872%	6	One	16306	16.306%
105/211	49.763%	6	Two	5436	5.436%
112/211	53.080%	8	One	10969	10.969%
104/211	49.289%	8	Two	3512	3.512%
108/211	51.185%	10	One	5773	5.773%
100/211	47.393%	10	Two	1555	1.555%

From examining the activation weights of the Zero-day files from Experiment 2 - Hybrid Attack (8.6%), which had 202 successful attacks out of 541 attempts, it was found that 172 of the successful attacks had six neurons outside two MADs and 10 successful attacks had eight neurons outside two MADs from the benign activation weights average. The results are shown in Table 50

Table 50 – Hybrid Attack (8.6%) Defence Results

Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Benign File Average	Suspicious Benign FPR
200/202	99.090%	6	One	16225	16.225%
172/202	85.149%	6	Two	5191	5.191%
199/202	98.515%	8	One	10394	10.394%
10/202	4.950%	8	Two	3375	3.375%
7/202	3.465%	10	One	5446	5.446%
2/202	0.990%	10	Two	1516	1.516%

From examining the activation weights of the Zero-day files from Experiment 3 - Hybrid Attack Randomised (8.6%), which had 184 successful attacks out of 541 attempts, it was found that 155 successful attacks had six neurons outside two MADs and 145 successful attacks had eight neurons outside two MADs from the benign activation weights average. The results are shown in Table 51.

Table 51 - Hybrid Attack Randomised (8.6%) Defence Results

Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Benign File Average	Suspicious Benign FPR
175/184	95.109%	6	One	15973	15.973%
155/184	84.240%	6	Two	5267	5.267%
170/184	92.391%	8	One	10568	10.568%
145/184	78.804%	8	Two	3394	3.394%
142/184	77.174%	10	One	5445	5.445%
113/184	61.413%	10	Two	1517	1.517%

From examining the activation weights of the Zero-day files from Experiment 4 - Combined Attack Randomised (15%), which had 436 successful attacks out of 541 attempts, it was found that 376 successful attacks had six neurons outside two MADs and 258 successful attacks had eight neurons outside two MADs from the benign activation weights average. The results are shown in Table 52.

Table 52 - Hybrid Attack Randomised (15%) Defence Results

Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Benign File Average	Suspicious Benign FPR
422/436	96.789%	6	One	15618	15.618%
376/436	86.239%	6	Two	6539	6.539%
408/436	93.578%	8	One	10919	10.919%
258/436	59.174%	8	Two	4613	4.613%
211/436	48.394%	10	One	6478	6.748%
33/436	7.569%	10	Two	2776	2.776%

From examining the activation weights of the Zero-day files from Experiment 5 - Combined Attack Randomised (10%), which had 135 successful attacks out of 531 attempts, it was found that 127 successful attacks had six neurons outside two MADs and 104 successful attacks had eight neurons outside two MADs from the benign activation weights average. The results are shown in Table 53.

Table 53 - Combined Attack Randomised (10%) Defence Results

Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Benign File Average	Suspicious Benign FPR
131/135	97.037%	6	One	15618	15.618%
127/135	94.074%	6	Two	5383	5.383%
131/135	97.037%	8	One	10601	10.601%
104/135	77.037%	8	Two	3360	3.360%
126/135	93.333%	10	One	5474	5.474%
1/135	0.741%	10	Two	1181	1.181%

For each of the tested defences, the clean test dataset from EMBER was used to calculate the average activation weights of each neuron from the last hidden layer. It was not expected that the average activation weights would be influenced in any way if the test dataset happened to also be poisoned with the adversarial features. To verify that the MAD defence can be performed using either a clean or poisoned dataset, 10% of the clean test dataset was injected with the adversarial features and the defence was repeated. Out of the 100,000 benign files in the clean test dataset, only 1,818 files contained all four target import libraries. As there was not enough test files to inject with adversarial features following the process from the combined poisoning attack, the injection process was modified to inject the import library and import functions into a random selection of 10,000 benign test files. The results from the defence experiments are shown in Table 54 with the results being very similar to the MAD defence on the clean dataset. The results from the experiment indicate that a known clean dataset is not required for the MAD defence, which is beneficial as it is not possible to know for certain if a dataset has been poisoned or not.

Table 54 - MAD Defence Experiment - Poisoned Test Data

Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Benign File Average	Suspicious Benign FPR
133/135	98.519%	6	One	15981	15.981%
127/135	94.074%	6	Two	4789	4.789%
133/135	98.519%	8	One	10864	10.864%
102/135	75.556%	8	Two	2896	2.896%
126/135	93.333%	10	One	5614	5.614%
1/135	0.741%	10	Two	967	0.967%

The examination of the Zero-day activation weights was compared to the average benign and malicious test files from the poisoned model. The defence does not require a clean model for use in any type of comparison, which is beneficial for mitigating the poisoning attacks as it is not possible to know for certain if any model was trained on clean data.

The tested defensive strategies are related to RQ2 and hypothesis nine which are restated below:

RQ2: Can adversarial attacks against machine learning based malware detection be mitigated?

a. Can targeted adversarial attacks be detected at test time?

H₉. The targeted adversarial attack can be prevented at test time.

It was possible to identify misclassified Zero-day files by examining the activation weights from the last hidden layer of the poisoned model. Each poisoned model from every test in the set of MLP experiments was analysed to see if it was possible to identify a misclassified Zero-day malicious file by examining outliers in 6, 8 and 10 neurons. The outliers were first examined by using one standard deviation from the mean of the benign activation weights from the training set. After examining the activation weights from the poisoned training dataset (50,000 samples), the test dataset set of 200,000 samples was evaluated to compare the results of the defence from both a poisoned and clean dataset.

When performing the additional examination of the 200,000 file test dataset, it was found that the variance of the activation weights in the 200,000 file data set was greater than the variance in the smaller data set which prevented the original defence method from working. To identify outliers within the test dataset, the mean absolute deviation (MAD) was used instead of the standard deviation. The MAD was calculated from the absolute values of the neuron's activation weights as half of the neurons were negative values.

The MAD examinations tested to see if the activation weights of the Zero-day neurons were outside of one and two MADs for 6, 8, and 10 neurons. In every set of experiments, it was possible to identify misclassified Zero-day malicious files with varying degrees of accuracy. In Experiment 5 Combined Attack Randomised (10%) Defence, the two MAD outliers for ten neurons had the worst performance with only one successful identification of a Zero-day malicious file.

The 100,000 benign test data files were also examined in each defence experiment to identify how many legitimate benign files would be categorised as suspicious. Unsurprisingly, the two MAD 10 neuron examination had the least number of false positives in the benign test data, the stricter the parameters for identifying suspicious files the less accurate the defence becomes. From all of the experiments performed, Experiment 5 Combined Attack Randomised (10%) would be the most practical to perform in a live environment, which had the best defence results (high detection success with low FPR) from 6 and 8 neurons outside of two MADs from the benign activation weight

average. The results from the defence experiments support H_9 , in that it is possible to detect a targeted adversarial poisoning attack at test time, an approach which has not been explored in past research. An in-depth analysis of the results from the experiments undertaken in this thesis is discussed in the following chapter.

4.7 Summary

In this chapter, a set of preliminary experiments were undertaken to identify the malware files which could represent Zero-day malware in the targeted adversarial poisoning attack experiments, and what level of efficacy was identified from selecting the imports section of Windows PE32 files as the target feature space of the adversarial poisoning attacks. In the main experiment section, targeted adversarial poisoning attacks were tested on both GBDT and ANN trained ML models. The aim of the targeted adversarial poisoning attacks was to allow for a Zero-day malware file, which was correctly identified as malicious using the clean ML models, to bypass detection which the general efficacy of the ML model was maintained. The poisoning attacks were followed by exploring different strategies for defending against the targeted Zero-day poisoning attacks.

The first two strategies, heatmaps and feature importance, failed in preventing the targeted adversarial attacks from succeeding, but they did provide insight for how the third, successful defence, was to be developed. In the final defence strategy, the activation weights of each Zero-day file from each successful attack were examined, and if found to be outside a normal range, the Zero-day file was flagged as suspicious. The normal range was defined as being within two mean absolute deviations of the average benign activation weights, which were calculated from the last hidden layer of the benign files from the 200,000 file test dataset. The average benign activation weights from the last hidden layer were selected as the last hidden layer contains the encoded data from the previous layers, and as the target attack is aiming to misclassify the Zero-day file into the benign class, if it falls outside the normal range, it is to be considered as suspicious.

From the defence experiments, it was found that a targeted Zero-day poisoning attack can be defended against at test time by examining the activation weights from the last hidden layer. The MAD defence had the least success in defending against the individual attack experiments with the greatest success rate of 56.872% but resulting in a 16.306% FPR when tested on the clean benign files from the test dataset. The best method for detecting suspicious files was if six or more neurons had activation weights outside two MADs from the benign average, which for experiments two, three, four and five achieved success rates of 84.313%, 84.946%, 86.073% and 97.810% with FPRs of 5.191%, 5.267%, 6.539% and 4.629%.

5 Results

This chapter discusses the results obtained from the experiments in chapter 4, the preliminary experiments that explored reducing the general efficacy of the ML model, the different targeted adversarial Zero-day poisoning attacks which were performed on both GBDT and MLP models, and finally the results obtained from the developed defensive technique to detect the Zero-day malware file at test time.

5.1 Preliminary Results

The preliminary experiments in 1.1 were performed to identify if the imported features of the training dataset were a suitable target for performing the Zero-day poisoning attack and if so, what parameters were to be selected for performing the attack. In the preliminary experiments, it was found that using the total feature space of the benign import features as a baseline for injecting the adversarial features was not a suitable approach for performing the adversarial poisoning attack. When selecting for 5% of the benign feature space of the import features, it was found that that amount of injection was never achieved with the selected features. In each experiment the total amount of benign files which contained the targeted adversarial feature were injected with the adversarial features. It would not be possible in a targeted black-box attack scenario to inject every benign training file with adversarial features, so this approach was not selected for use in the targeted Zero-day adversarial attack experiments.

The second stage of the preliminary experiments (section 4.4.2), used varying percentages of the benign training files which contained the targeted import library. The attacks were performed in steps of 5%, starting at 5% and finishing at 100%. The results from the majority of the experiments did not significantly change the FNR of the model until the last experiment which injected poison features into every instance of a benign training file that contained the target import library. The threshold did increase throughout the majority of the experiments, indicating that the presence of adversarial features influenced the training of the model, but not at a significant level until the final stages of the experiments, as shown in Table 55.

Table 55 - Kernel32.dll - Last Three Attack Results from each Preliminary Experiment

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC %	Detection Rate 1%
manual_kernel32_5_90	0.999	2.826	0.761	99.847	97.174
manual_kernel32_5_95	0.994	4.389	0.854	99.768	95.611
manual_kernel32_5_100	0.986	13.379	0.980	99.291	86.621
manual_kernel32_10_90	0.997	3.370	0.804	99.815	96.630
manual_kernel32_10_95	1.000	4.786	0.881	99.754	95.214
manual_kernel32_10_100	0.992	17.118	0.989	99.188	82.882
random_kernel32_5_90	0.992	3.219	0.877	99.838	96.781
random_kernel32_5_95	0.989	5.344	0.941	99.753	94.656
random_kernel32_5_100	0.000	100.000	1.000	98.143	0.000
random_kernel32_10_90	0.989	3.513	0.892	99.821	96.487
random_kernel32_10_95	0.990	4.681	0.945	99.766	95.319
random_kernel32_10_100	0.000	100.000	1.000	97.867	0.000

The hypotheses tested in the preliminary experiments were H_1 , H_2 , H_3 , and H_4 which are stated below and shown in Table 56.

H_1 . A manual selection of features can be used for a successful adversarial poisoning attack.

H_2 . A random selection of features can be used for a successful adversarial poisoning attack.

H_3 . No more than 5% of the benign feature space is required to be poisoned to reduce the general efficacy of the model.

H_4 . No more than 10% of the training data is required to be poisoned to reduce the general efficacy of the model.

Table 56 - H_1 - H_4 Experiment relationship and Results

Hypothesis	Related Experiments	Supported
H_1	First Preliminary Set	Yes
H_2	Second Preliminary Set	Yes
H_3	All	Yes
H_4	All	No

The only hypothesis which was not supported by the results from any of the preliminary experiments was H_4 . None of the experiments were able to reduce the general efficacy of the model at an injection rate of 10% benign training files containing the targeted import library.

It was shown that it is possible to reduce the overall availability of the model from both a manual and random selection of adversarial features which supports H_1 and H_2 , the overall amount of training files

which needed to be injected when using the chosen sample set of adversarial features was significantly large (100% of the target benign files). The high percentage of poisoning was likely due to the small number of features chosen for injection, as the imports section of Windows PE32 files in EMBER comprises the majority of the feature vectors used for training the model.

H₃ was supported from the experiments as 5% of the benign feature space of the training data being injected with adversarial features was never achieved using the target manual selection of adversarial features. In the benign feature space experiments, it was found that every instance of benign training files which contained the target import library were injected with the adversarial features, the same as the 100% injection attack performed in the other set of preliminary experiments. The target import library “msvcrt.dll” was the only experiment to not reduce the overall availability of the model after injecting the adversarial features into every benign training file. The import library “msvcrt.dll” was not very common in the EMBER dataset, therefore, the results from the general efficacy attack experiments were as expected.

5.2 Targeted Attack Results (GBDT)

The GBDT model which is supplied in the EMBER framework was the first model tested in the targeted adversarial poisoning experiments. The first set of experiments performed used a subset of 50 Zero-day files out of the chosen 541 Zero-day files obtained from VirusShare. These experiments tested different percentages of injection of the total benign training files. The selected percentages were 5%, 10%, 15%, 20% and 25%. Unlike the preceding preliminary experiments which continued to a maximum of 100% injection, it was not intended that the targeted adversarial poisoning attacks should affect the overall availability of the model or increase the threshold for detecting malicious files too far from the original clean model. The aim of the experiments was to see how well the poisoning attacks would perform, without going too far beyond a reasonable level of poisoning. The results from the experiments are shown in Table 57.

Table 57 - Targeted Adversarial Poisoning Attack (Individual Approach) - GBDT Results

Test	Total Successful Attacks	Average Score	Median Score	Average Threshold	Median Threshold
5%	0	0.901	0.918	0.539	0.538
10%	2	0.813	0.851	0.560	0.561
15%	9	0.748	0.832	0.586	0.589
20%	14	0.696	0.765	0.619	0.624
25%	19	0.645	0.713	0.657	0.664

The second experiment, which was called the combined attack, only injected the adversarial features into benign training files which contained all four target import libraries. The combined attack experiment reduced the amount of available target benign training files for poisoning but was designed to create a pattern within the benign class, which when triggered by the Zero-day malware test file, would shift the file's classification from benign to malicious. The results from the combined attack experiments are shown in Table 58. The difference between the two experiments is that in the first experiment, adversarial features were injected on a first come basis, which would lead to larger clusters of poisoned data in the benign training set, while the second attack randomised the selection of target benign files to poison. In the second experiment, the same random seed was used in each attack to provide a uniform way of comparing the efficacy of each attack.

Table 58 - Targeted Adversarial Poisoning Attack (Combined Approach) - GBDT Results

Test	Total Successful Attacks	Average Score	Median Score	Average Threshold	Median Threshold
First Experiment	328/543	0.519	0.538	0.607	0.608
Second Experiment	312/543	0.531	0.551	0.606	0.606

The targeted Zero-day poisoning experiments are related to research question one and hypotheses five and six, which are restated below.

RQ1: Can adversarial attacks against machine learning based malware detectors increase the likelihood of unknown-unknown malware samples bypassing detection?

H₅. No more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.

As it was shown in Table 57, neither the 5% attack or the 10% attack had any significant number of successful targeted Zero-day poisoning attacks, the 5% test had zero successful attacks and the 10% attack only had two. The individual attack approach was not expected to have any significant number of successful attacks from how the attack was performed but it was assumed that there would be more successful attacks than was shown. Neither of the hypotheses were supported by the results obtained from the first individual attack experiments.

As it was shown in Table 58, the hybrid attack experiments, which used an 8.6% injection of the total benign training files, achieved a success rate of 60.40% (326/541 successful attacks) for the first set of experiments, while the second set of experiments had a success rate of 57.45% (310/541 successful attacks). The results from both sets of experiments support H_5 , in that successful targeted adversarial poisoning attacks can be achieved with poisoning less than 10% of the total benign training files.

5.3 Targeted Attack Results (MLP)

The second ML algorithm used to test the targeted Zero-day adversarial poisoning attack was a basic Multi-Layer Perceptron (MLP). The architecture of the MLP was adapted from an example used by Endgame, the creators of EMBER, to keep a level of consistency between the tests instead of creating and calibrating a new MLP model. The hidden layers of the MLP were modified from the original Endgame example, with the main addition being an extra layer of ten neurons. The ten-neuron layer was the last hidden layer of the model and was included as the idea of examining the last hidden layer to identify the targeted Zero-day poisoning attack had already been considered as a possible approach for developing a defence.

The training data was comprised of 50,000 samples (25,000 benign and 25,000 malicious) which were extracted from the EMBER2017 dataset. The dataset was split evenly between malicious and benign files to follow the format of the original EMBER2017 dataset, which contained 300,000 malicious and 300,000 benign files. The benign files were extracted to contain enough files to generate the adversarial examples containing the four chosen import libraries. These adversarial example files were extracted on a first come basis and were deliberately not selected from a list of successful poisoning attacks from the GBDT experiments described in section 4.5.1.1, so as to not directly influence the following experiments.

Five sets of experiments were carried out to examine how the poisoning attack influenced the classification of the MLP model. In the first experiment, the adversarial features were injected separately into benign files in the same way as the individual attack mentioned in the GBDT experiments. The injection was performed at a rate of 25% of the total benign files in the training data set. For each of the experiments in the individual attack, the poisoned training data generated were not randomised before training the ML model. The attack had 213/541 Zero-day files misclassified as benign, a success rate of 39.22% while not negatively influencing the general efficacy of the model.

In the second set of experiments, a hybrid injection attack was performed at 8.6% of the total benign files. The dataset was generated to contain 15% of the benign files available for injection (they contained all four chosen import libraries). The second experiment also did not randomise the poisoned training data, which may influence the way the attack performs as the original generation of the training set created some clusters of benign files which were available for injection. The attack

had 202/541 Zero-day files misclassified as benign, a success rate of 37.338%, which is less than the individual attack and contrary to the results from the individual and combined GBDT attacks.

The third and fourth sets of experiments both randomised the poisoned training data before generating the model. The randomisation of training data was only performed to reduce the clusters of benign data which exist from the original dataset, the seed was held constant across these experiments for reproducibility. The third test was performed at 8.6% and the fourth test at 16.5% of the benign file space. The third test had 184/541 (34.011%) Zero-day files misclassified as benign while the fourth test had 436/541 (80.591%).

The general efficacy of the model (as measured by the accuracy and threshold) in the third set of experiments was the same as the previous two sets of experiments. In contrast, in the fourth set of experiments the false negative rate and the threshold drastically increased. The increase in successful attacks in the fourth set of experiments was due to the degradation of the general efficacy, as shown in Table 59, the average adversarial score was 0.863 while the average threshold was 0.984, which shows that the poisoning attack did not reduce the overall score of the Zero-day files in a significant way, instead the overall threshold was increased, which allowed for the Zero-day files to be misclassified as benign.

The combined attack experiments achieved a success rate of 24.954%, with 135 out of 541 successful attacks. The combined attack experiments had fewer successful attacks compared to both hybrid attacks which injected adversarial features into a smaller amount of benign training files. It is likely that the increased number of training files poisoned with adversarial features from 'user32.dll' influenced the success of the hybrid poisoning attacks.

Table 59 - MLP Targeted Adversarial Attack Results

Test Name	Total Successful Attacks	Average Score	Median Score	Average Threshold	Median Threshold	Average FNR
Individual Attack (23%)	211/541	0.683	0.986	0.856	0.858	4.040
Hybrid Attack (8.6%)	202/541	0.751	0.998	0.834	0.835	3.994
Hybrid Attack Randomised (8.6%)	184/541	0.823	0.997	0.811	0.813	3.358
Hybrid Attack (17%)	436/541	0.866	0.963	0.984	0.988	18.569
Combined Attack (10%)	135/541	0.914	0.993	0.837	0.836	4.049

The results from the targeted Zero-day poisoning experiments show that it is possible to increase the likelihood of unknown-unknown malware samples bypassing detection from ML trained malware detection models. In both the GBDT and MLP trained models' successful attempts to bypass classification through poisoning of the benign training set was achieved. The experiments did require up to ten percent of the benign training data samples to be infected with adversarial features but the process of poisoning the model was performed in a black-box scenario, which reduced the ability to identify which features would contribute the most to the model's classification and to target those features specifically. From the feature analysis paper by Saxe and Berlin (2015), it was found that the imports section when used by itself to identify patterns and correctly classify malicious files, performed the worst compared to the four other areas that were tested. In the experiments the imports section was chosen as it compromised the majority of the input feature vectors and in a practical sense, adding an import function to a binary PE32 file is a simple way to perform an adversarial

poisoning attack. In future work other areas of the binary PE32 file would be tested to identify which area is best suited for manipulation to generate a successful targeted adversarial poisoning attack.

5.4 Defence Results

The defensive method developed in this thesis examines the activation weights of files at test time from the last hidden layer of the MLP ML model to identify anomalous behaviour in the weights. In a successfully trained model, it would be expected that the activation weights would be clustered around the average of a correct class for each of the neurons in the last hidden layer. Only the last hidden layer needs to be examined as all the previous data is encoded in this layer. As normal files which are either benign or malicious sit close to the average activation weight when examining a clean model, it would be obvious that something has gone awry when a tested file sits between the activation weights of both classes for the majority of the neurons, as shown in the example below:

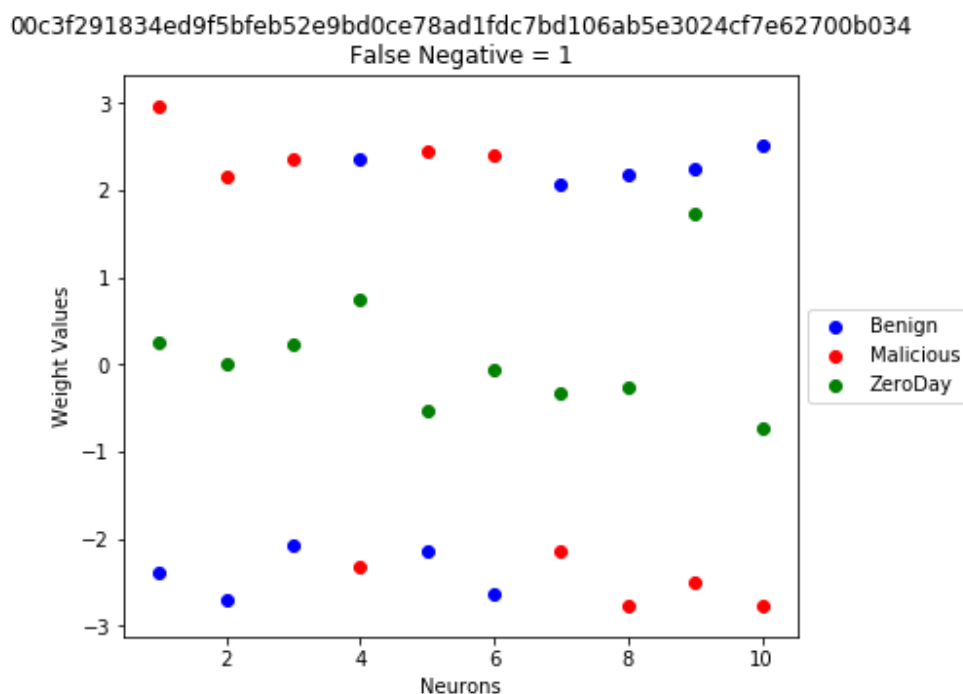


Figure 5.1 - ZeroDay False Negative Example 1

In Figure 5.1 - ZeroDay False Negative Example 1, the red and blue dots represent the average activation weights of the malicious and benign test files from EMBER's 200,000 file test dataset, respectively. The green dots are the activation weights of the target Zero-day file which was used as the basis for extracting import functions and injecting them throughout the benign training dataset. In a clean (non-poisoned) model, the green dots would be closer to the red dots (because Zero-day is a sub-class of Malicious), but due to the adversarial poisoning, the activation weights have shifted away from their original malicious class and moved closer towards the benign class. The activation weights

have not shifted enough to the benign class to make the Zero-day file appear as the legitimate benign files do, but the shift in the activation weights has been significant enough for the malicious Zero-day file to be misclassified as benign. To defend against targeted Zero-day poisoning attacks, a test was proposed to evaluate the activation weights at test time. The heuristic used in this test was if it was found that a file which was classified as benign, had eight or more activation weights that were further than one standard deviation away from the average activation weight of the neuron, then the file would be considered to be suspicious and should be quarantined for further examination which could be performed by dynamic analysis or any other methods for malware detection. Three sets of experiments analysing six, eight, and ten neurons of the activation weights at test time were performed, and the eight neuron experiment was found to have the most successful results.

From the 184 successful attacks from Experiment 3 shown in Table 59, 182 (97.85%) of the Zero-day files had eight or more activations weights outside of one standard deviation from their corresponding benign neuron activation weight value. Using the first successful attack as an example, the benign activation weights from the training dataset were examined and a total of 1,553 from 25,000 (6.21%), which is near the percentage of adversarial examples in the benign training data, were outside the range, and would be regarded as suspicious. To determine if the activation weights were outside the range in the benign training data was due to the poisoning attack, the activation weights of the benign training files from the clean model were examined, and 1,817 files (7.26%) had eight or more activation weights outside of the range, which is larger than the poison example. Which indicates that it is not the poisoning which is responsible for genuine benign files to appear as suspicious in accordance with the defence, but that legitimate variance in benign files exists which will generate false positives for the defensive strategy.

After using the benign activation weights of the 25,000 files used to train the model, additional examinations were performed using the 100,000 benign data files from the original EMBER2017 dataset. When calculating the averages and standard deviation of the activation weights from the 100,000 benign test files for each poisoned model, it was found that the average of the activation weights were similar across each experiment but the standard deviation was significantly different. From using a larger dataset, a greater amount of variance was introduced which included a significant number of outliers which prevented the previously successful defensive technique from functioning correctly.

To remove the issue of the outliers in the dataset rendering the defensive technique unusable, another method was examined to identify if it was possible to detect suspicious benign classifications from examining the activation weights at the last hidden layer during test time. The method which was chosen for use is known as Mean Absolute Deviation (MAD), which is another technique for

identifying anomalies in data without the present outliers in the dataset influencing the standard deviations to such a significant degree.

For each of the five sets of experiments performed on the MLP model, two defensive strategies were tested using the MAD. The defensive strategies followed the same formula as the original defence i.e., examining the activation weights from the last hidden layer of the target Zero-day file, but the MAD defence also tested to see if the activation weights of the neurons were outside of two MADs, instead of the single standard deviation which was originally used. The two MADs were also used as the general rule for identifying outliers is if they are two standard deviations away from the mean. For the fifth set of experiments, an additional set of defensive sets was performed using activation weights obtained from a poisoned version of the test dataset. In the poisoned set, ten percent of the benign files were injected with the adversarial features. The defence using the poisoned weights was performed to verify that the MAD defence does not require a clean dataset to generate the average benign activation weights. The results from each MAD defence are shown in Table 60, Table 61, and Table 62.

Table 60 – MAD Defence Results Six Neurons

Experiment	Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Suspicious Benign FPR
One	120/211	56.87%	6	One	16.306%
	105/211	49.76%	6	Two	5.436%
Two	200/202	99.09%	6	One	16.225%
	172/202	85.15%	6	Two	5.191%
Three	175/184	95.11%	6	One	15.973%
	155/184	84.24%	6	Two	5.267%
Four	422/436	96.79%	6	One	15.618%
	376/436	86.24%	6	Two	6.539%
Five	131/135	97.04%	6	One	15.618%
	127/135	94.07%	6	Two	5.383%
Five (Poisoned Test Data)	133/135	98.52%	6	One	15.981%
	127/135	94.07%	6	Two	4.789%

Table 61 - MAD Defence Results Eight Neurons

Experiment	Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Suspicious Benign FPR
One	112/211	53.08%	8	One	10.969%
	104/211	49.29%	8	Two	3.512%
Two	199/202	98.52%	8	One	10.394%
	10/202	4.95%	8	Two	3.375%
Three	170/184	92.39%	8	One	10.568%
	145/184	78.80%	8	Two	3.394%
Four	408/436	93.58%	8	One	10.919%
	258/436	59.17%	8	Two	4.613%
Five	131/135	97.04%	8	One	10.601%
	104/135	77.04%	8	Two	3.360%
Five (Poisoned Test Data)	133/135	98.52%	8	One	10.864%
	102/135	75.56%	8	Two	2.896%

Table 62 - MAD Defence Results Ten Neurons

Experiment	Successful Defence	Success Percent	Neurons	Mean Absolute Deviations	Suspicious Benign FPR
One	108/211	51.19%	10	One	5.773%
	100/211	47.39%	10	Two	1.555%
Two	7/202	3.47%	10	One	5.446%
	2/202	0.99%	10	Two	1.516%
Three	142/184	77.17%	10	One	5.445%
	113/184	61.41%	10	Two	1.517%
Four	211/436	48.39%	10	One	6.478%
	33/436	7.57%	10	Two	2.776%
Five	126/135	93.33%	10	One	5.474%
	1/135	0.74%	10	Two	1.181%
Five (Poisoned Test Data)	126/135	93.33%	10	One	5.614%
	1/135	0.74%	10	Two	0.967%

The tested defensive strategies are related to RQ2 and hypothesis nine which are restated below:

RQ2: Can adversarial attacks against machine learning based malware detection be mitigated?

a. Can targeted adversarial attacks be detected at test time?

H₆. The targeted adversarial attack can be prevented at test time.

It was possible to identify misclassified Zero-day files by examining the activation weights from the last hidden layer of the poisoned model. Each poisoned model from every experiment in the set of MLP experiments was tested to see if it was possible to identify a misclassified Zero-day malicious file by examining outliers in 6, 8 and 10 neurons. The outliers were first examined by using one standard deviation from the mean of the benign activation weights from the training set. An additional examination was performed using the 200,000 file test dataset from EMBER2017, as the training dataset contained poisoned samples whereas the test dataset was clean. When performing an additional examination on the 200,000 file test dataset, it was found that the variance of the activation weights in the 200,000 file data set was greater than the variance in the smaller data set variation of the activation weights from the larger dataset increased the standard deviation spread which prevented the original defence method from functioning correctly. As an alternative measure to identify outliers within the test dataset, the mean absolute deviation (described in section 4.6.3.2) was used instead of the standard deviation. The mean absolute deviation was calculated from the absolute values of the neuron's activation weights as half of the neurons were negative values.

The MAD examinations tested to see if the activation weights of the Zero-day neurons were outside of one and two MADs for 6, 8, and 10 neurons. In every set of experiments, it was possible to identify misclassified Zero-day malicious files with varying degrees of accuracy. In Experiment 5 Combined Attack (10%) Defence, the two MAD outliers for ten neurons had the worst performance with only one successful identification of a Zero-day malicious file.

The 100,000 benign test data files were also examined in each defence experiment to identify how many legitimate benign files would be categorised as suspicious. Unsurprisingly, the two MAD 10 neuron examination had the least number of false positives in the benign test data, as the stricter the parameters for identifying suspicious files, the less accurate the defence becomes. From all of the experiments performed, Experiment 5 Combined Attack Randomised (10%) would be the most practical to perform in a live environment, from examining Experiments 5 defence results, the tests for 6 and 8 neurons outside of two MADs from the benign activation weight average provided the best defence success (94.07% and 77.04%) along with low percentages of false positives (5.383% and 3.360%). The results from the defence experiments support H₉, in that it is possible to detect a targeted adversarial poisoning attack at test time, an approach which had not been explored in past research.

5.5 Defence Contribution

The MAD defence proposed and tested in this thesis differs from the current adversarial poisoning defences in that it does not aim to use some form of outlier detection to detect the poisoned samples within the training dataset and remove them, but instead aims to detect suspicious files at test time by their activation weight values and quarantine the file for further analysis. Unlike most existing training defences which require a clean dataset to compare against the poisoned dataset to identify anomalous clusters, the MAD defence can be performed successfully using either clean or poisoned data to generate the activation weight values. Being able to defend against poisoning attacks without having a confirmed clean dataset to use for comparison is beneficial as it is not possible to know for certain if a dataset is clean or poisoned. The novelty of defending against adversarial attacks at test time provides an extra layer of security to defend against poisoning attacks. The adversarial defences which remove poisoned data from the training set should be utilised to protect against poisoning attacks in conjunction with the test time MAD defence to provide the optimal level of defence.

5.6 Summary

In this chapter, an overview of the different poisoning experiments was provided along with an analysis of the results and how they relate to the research questions and hypotheses. From the preliminary poisoning experiments, it was found that the results support that both a manual and random selection of adversarial features can be used in a successful poisoning attack (H_1 and H_2) and that no more than 5% of the benign feature space is required to be poisoned for a successful attack to occur (H_3). The results from the preliminary experiment did not support H_4 , as more than 10% of the benign training files were required to be poisoned for the attacks to succeed.

From the main set of experiments, it was found that it was possible for a targeted adversarial poisoning attack to succeed without poisoning more than 10% of the benign training data (H_5). The main experiments were performed using different poisoning methods to identify the importance of adversarial features and data positioning in relation to attack success. It was found that if the poisoned training files clustered closer together, more attacks would succeed.

In addition to the poisoning experiments, an analysis of the results from the MAD defence strategy was provided and it was found that the overall most successful defence was six or more neurons outside of two MADs from the target benign average, which for experiments two, three, four and five achieved success rates of 84.313%, 84.946%, 86.073% and 97.810% with FPRs of 5.191%, 5.267%, 6.539% and 4.629%.

6 Discussion

In this chapter, a discussion of the attacks, defences and their relation to the proposed research questions is provided. The chapter starts with a review of the preliminary experiments, which provided the basis for the targeted adversarial attacks, as the preliminary experiments identified what type of features to be used in the poisoning attacks and that they could be randomly selected without the need to perform any feature importance analysis. The chapter continues with a discussion of the targeted poisoning attack, and how the attack differed from the watermark attack which also used the EMBER dataset. The next section covers the types of defences which have been developed to prevent adversarial poisoning attacks and how the proposed MAD defence in this thesis differs from the other defences and provides a new way to defend against targeted adversarial poisoning attacks. The final section outlines each research question and hypothesis and examines the results from their related experiments which either support or reject the hypothesis.

6.1 Adversarial Attack Discussion

A selection of different types of attacks were performed during the experimentation stage of the thesis. The first attacks were executed to identify if using the import library section as the basis of the poisoning attack would be successful. The second set of attacks were performed to see if it was possible for a targeted Zero-day malware file, which was originally classified correctly, to be misclassified as a benign file after injecting features from the import section of the targeted Zero-day into the benign training files import section. For the targeted adversarial poisoning attacks, CNN and RNN models were also evaluated to provide a comparison between CNN, RNN, and MLP models. All the trained CNN models produced a no-skill model which did not exceed 50% accuracy. None of the RNN models completed training stage, due to the hardware limitations of the workstation. As restrictions in were in place to prevent the spread of covid-19, it was not possible to gain access to more powerful equipment which could train the RNN model, and as such, only MLP models were used throughout the rest of the experimentation stage.

The general efficacy attack performed in the preliminary experiments is similar to the various adversarial poisoning attacks which have performed on a variety of ML models, such as the SPAM poisoning attack performed by Nelson et al. (2008). The purpose of the preliminary experiments was to first validate existing research and second, to identify possible parameters for performing the targeted Zero-day poisoning attack. The primary aim of the preliminary experiments was to identify if injecting adversarial features into the import section of the benign training data files would influence the classification of the trained model, and the secondary aim of the preliminary experiments was to identify if there was any significant difference from using either a manual or random selection of features. The manual selections of features were chosen by analysing the feature count from the benign training file import library functions, and from the malicious Zero-day files obtained from

VirusShare. The random selections of features were randomly taken from the total pool of import library functions in both the benign EMBER set and malicious VirusShare set.

In the general efficacy attack experiments, each chosen import library ('kernel32.dll', 'advapi32.dll', 'user32.dll', 'shell32.dll', and 'msvcrt.dll') was used separately when performing the adversarial poisoning attacks. The adversarial poisoning attacks were performed at different percentages of the total benign train data files. The first set of experiments were performed using the feature space of the import library functions as the basis of the poisoning attack, with the first and only set of experiments using 5% of the total feature space as the maximum injection percentage. The feature space injection attacks were only performed at 5% as that percentage was not achieved with every instance of benign training files which contained the target import library being injected with the adversarial features.

The other set of preliminary experiments used the percentage of benign training files which contained the target import function as the basis of the attacks. The first experiment started at 5% and continued in steps of five until 100% of the available benign training files were injected with the adversarial features, the last attack at 100% being the same as the feature space attack.

The results from the experiments indicated that using the import section of the binary PE32 files as the target for the adversarial poisoning attacks would be suitable for influencing the classification of the model when performed the targeted Zero-day adversarial poisoning attack. The import section of binary PE32 files was found to be the least important when identifying patterns to distinguish between malicious and benign files in the paper by Saxe and Berlin (2015). The results from the paper were used by the developers of EMBER when performing the feature engineering stage of their framework. It is interesting to see that the field with the least contribution to the classification of malicious and benign files can be manipulated to allow for a targeted Zero-day file to bypass detection by being misclassified as benign, when it would otherwise be classified correctly from the same but clean dataset. In future work it would interesting to see how well the targeted adversarial poisoning attack would work when manipulating other sections of the binary PE32 file.

A similar attack to the targeted Zero-day poisoning attack was performed by Severi et al. (2020) which also used EMBER as the dataset for their attack which they called a watermark attack. The watermark attack was performed by manipulating a portion of the non-hashed input vectors of the benign files in the EMBER dataset to create a pattern that would classify any tested with the same watermark as benign.

The watermark attack used SHAP values (see section 2.7.1.8) to identify the contribution of each vector in the 2351-dimensional input vector and then selected the vectors with the greatest contribution as the target values for the attack. The watermark attack only selected features which were not hashed in the feature engineering phase of generating the EMBER model. Features in EMBER which are hashed are a harder target for creating a watermark as the majority of the hashed

features are generated from a series of features, which would mean that the majority of features which contribute to the hash would need to be manipulated to create the watermark. Only using features which are not hashed reduces the difficulty in creating the watermark. It would be interesting to see how well the watermark attack would work if the features which were manipulated were computed into several hashes.

The targeted Zero-day adversarial poisoning attacks performed in this thesis did not rely on feature analysis from a surrogate model to identify which features contributed the most to the classification as the watermark attack did. Instead, the targeted Zero-day adversarial poisoning attacks manipulated the imports section of the benign training files, which unlike the watermark attack, was a section which was computed into a hash digest of 1280-bits. The imports section was chosen as it was used to compute the majority of the input feature vectors used in EMBER, and it provided a straightforward avenue for manipulating the benign training files and performing the adversarial poisoning attacks.

6.2 Adversarial Defence Discussion

As stated in 3.2.2, the eighth project phase was the evaluation and discussion of the adversarial defensive strategy. The defensive strategies outlined in the papers by B. Chen et al. (2018) and Wang et al. (2019) follow a similar approach in that they both examine the activation weights from the last hidden layer of the neural network model. This is not surprising, as the last hidden layer contains all the encoded data from the previous layers during training. The previous defences were developed on multiclass image classification models, with the activation clustering defence also being used by Severi et al. (2020) for their backdoor watermark attack on EMBER. The MAD defence proposed in this thesis differs from the activation clustering defence in that it identifies poisoning attacks at test time, in contrast to identifying adversarial samples at the training stage and removing them.

6.2.1 Neuron Pruning

The neuron pruning defence proposed by Wang et al. (2019) identifies adversarial examples by examining the neurons at the last hidden layer for outliers in the top 1%. In the experiments performed by the authors, the multiclass classification models would have a small percentage of neurons which were 3-7 times as active for adversarial examples compared to clean examples. The poisoning attacks performed seem to overfit for the trigger in the adversarial examples, allowing for the target example to be misclassified in accordance with the attacker's plan. After identifying that a model has been compromised, the authors suggested pruning of neurons to remove the encoded pattern which allows for the poisoning attack to succeed. This type of defence would not be appropriate for the attacks performed in this thesis, as the targeted adversarial poisoning attacks performed do not create a pattern of neurons with activation weights 3-7 times the average for the target class, but instead shift the weights of the neurons to sit somewhere in the middle of both classes, but still closer to the target benign class when the attacks succeed.

6.2.2 Activation Clustering

In the activation clustering defence, B. Chen et al. (2018) were able to identify anomalous clusters (the backdoor) due to the backdoored image retaining the majority of the features from the source image. To fix the poisoned model, the authors relabel the identified adversarial examples as their original source class and retrain the model. The poisoned models from the targeted adversarial poisoning attacks in this thesis cannot be fixed by relabelling of the adversarial samples within the benign training data as the files are not mislabelled, they instead contain import functions found within both benign and malicious files which shift the classification of target Zero-day files. Retraining the model after identifying the adversarial examples would not be a permanent fix if the model is being updated periodically with new data obtained through the same vulnerable avenue (online gathering). For malware detection applications trained using machine learning, it would be beneficial to utilise a quarantine defence which isolates suspicious (adversarial) files for further examination. The quarantine approach would allow for the malware detection application to operate under the assumption that the model may have been poisoned by an attacker but can still function as intended.

The targeted adversarial poisoning attacks performed in the early stages of this research Wood and Johnstone (2020) has some similarities to the watermark poisoning attack subsequently reported by Severi et al. (2020). In both this thesis and the watermark paper, EMBER was chosen as the target dataset for testing the experiments and both a Gradient Boosted Decision Tree and Artificial Neural Network model were tested. In the watermark paper, the authors performed both white-box and black-box approaches, in this thesis, only a black-box approach performed. In the watermark paper, SHAP values were calculated to identify which features from EMBER's 2351 vector inputs contributed the most to either the benign or malicious classification. The identification of import feature vectors played a crucial part in identifying which features would be manipulated to create the target watermark. The watermark paper did not modify any features which would be later hashed by EMBER's feature hashing function, this reduced the total amount of available features from 2351 to 2316. In the black-box approach of this thesis, no precursor work was performed to identify the contribution of feature vectors to the model's classification. It was assumed to be unlikely that an attacker would have the underlying knowledge of the model's architecture but could make a safe assumption that the imports section of the file would be included in the feature space of the training data, due to either the model being trained using some form of feature analysis or being trained using the complete byte sequence of the file.

It would be interesting to test the watermark attack on a modified version of EMBER which converts the un-hashed values into a hash digest and see if there is any significant difference, first in the quality of the EMBER model and second in the efficacy of the watermark attack. If feature hashing prevents the use of SHAP values to determine which feature vectors contribute the most towards the model's

classification, would performing feature hashing on every feature prevent attackers from building substitute models to determine which features should be poisoned?

An experiment which has been left for future work, is to test the watermark attack against the test time activation weight defence produced in this thesis. It is possible that the defence would not be successful in identifying the watermark attack if the watermark is successful in shifting the activation weights closer to the benign average than the experiments in this thesis. As the experiments in this thesis did not use any precursor information for identifying which features of EMBER contributed the most to the model's classification, the attack succeeded instead by injecting enough adversarial features to shift the classification towards the benign classification, but not enough to appear as a legitimate benign file. The watermark attack however did use SHAP values, so it may be possible that the attack does in fact shift the activation weights of the target malicious file to be effectively camouflaged within the benign activation weights.

6.3 Adversarial Attack and Defence Comparison

From the discussion in the previous sections, an explicit comparison of the work undertaken in this thesis and the other adversarial attacks and defences discussed above can not be provided due to the clear differences in the experiment approaches. The adversarial attack which was most similar to the targeted adversarial poisoning attack conducted in this thesis was the watermark attack performed by Severi et al. (2020), as it also used the EMBER dataset and injected adversarial features into the benign feature space of the training data. The clear difference between the watermark attack and the targeted poisoning attack in this thesis was the pre-processing stage to identify salient information in the EMBER feature space performed in the watermark attack. The pre-processing stage in the watermark attack identified which features contributed the most to the model's classification, which provided the researchers with a direct avenue for attack. In comparison, no pre-processing was performed when undertaking the targeted adversarial poisoning attack in this thesis, and as such, a clear comparison of the results cannot be performed.

In the defensive approaches discussed in the previous sections, the adversarial examples in the training datasets were identified by evaluating the activation weights of the neurons at the last hidden layer of the poisoned model in comparison to a trusted clean model and removing the training files which were significantly outside of the normal distribution from the clean dataset. This type of defence works under two assumptions, first, a trusted clean dataset is available to generate a baseline for acceptable neuron activation weight values, and second, that the neuron activation weight values are easily distinguishable from the trusted clean dataset's normal distribution. The defence in this thesis operates under the opposite of the aforementioned assumptions, in that a trusted clean dataset is not required and that the neuron activation weights of poisoned data are not always easily distinguishable from clean data in the training dataset.

6.4 Research Question Discussion

In this thesis, two research questions were proposed, and a series of experiments were undertaken to gather the required information to answer the questions. In this section, the results gathered from the experiment stage are examined along with their corresponding research questions and hypotheses. The two research questions examining the attack and defence of machine learning models are restated below:

RQ1: Can adversarial attacks against machine learning based malware detectors increase the likelihood of unknown-unknown malware samples bypassing detection?

- a. What features are required to perform a successful poisoning attack?
- b. What percentage of poisoning is required to reduce the overall availability?
- c. What percentage of poisoning is required for a targeted attack to succeed?

RQ2: Can adversarial attacks against machine learning based malware detection be mitigated?

- a. Can targeted adversarial attacks be detected at test time?

From the two research questions, nine hypotheses were generated and tested throughout the experimentation stage. Each of the hypotheses is listed below, with an explanation from the information gathered during the experiments to either support or reject the hypothesis.

H₁. A manual selection of features can be used for a successful adversarial poisoning attack.

H₂. A random selection of features can be used for a successful adversarial poisoning attack.

The first set of experiments performed provided the building blocks for the later targeted backdoor attack. The experiments set out to perform a general efficacy attack against the machine learning model with different approaches to the feature selection used in the poisoning attack. The experiments in this thesis operated under the assumption that an attacker would not be able to identify the importance of features and would instead have to operate under the assumption that after poisoning enough of the training data, the model would shift classification towards the attacker's desired outcome. Therefore, this work is unlike the targeted backdoor watermark attack performed by Severi et al. (2020), in which precursor examination of the model was performed using both a white-box approach, and a black-box approach using a surrogate model to identify which features had the most significant influence on the models classification.

A manual selection of import library features was identified by analysing the training data and a selection of virus data to identify which features were more common among the virus set than the training set. This approach, if chosen for the targeted adversarial attack, would fall under the white-box category as the attacker would have complete access to the training data. The manual selection

approach was not intended to be used in the targeted adversarial attack but was used in the preliminary experiments to compare against the results obtained from the random selection attack.

After performing both the manual selection and random selection set of poisoning experiments, the results indicated that, EMBER is vulnerable to adversarial poisoning attacks targeting the general efficacy of the model. Further, there is no clear difference from using a manual selection of features or a random selection as the basis of the attack. From the results of both sets of experiments, it appeared that a random selection of adversarial features could be used in the future targeted attack.

H₃. No more than 5% of the benign training data feature space is required to be poisoned to reduce the general efficacy of the model.

H₄. No more than 10% of the benign training data files are required to be poisoned to reduce the general efficacy of the model.

In the general efficacy attacks, different percentages of poisoned training data were tested to gauge what level of injection would be required to succeed in performing a targeted adversarial attack. Only H₄ was not supported from the experiments, as for the majority of the experiments, the general efficacy of the model was only significantly affected when the adversarial poisoning had reached 100% of the benign training files which could be targeted in the attack.

H₅. No more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.

The targeted adversarial poisoning attacks which succeeded used 8.6% of the benign training data and 2.5% of the total training data. Both the GBDT and MLP models were vulnerable to the targeted adversarial poisoning attack, with the MLP models being more resilient. Less data was used to train the MLP models which may have been the reason behind the increased resilience. It is possible that training files which had a significant influence over the classification of the model and would have been targeted in the adversarial poisoning attack may not have been included in the MLP dataset. The MLP attacks may be replicated in the future using a more powerful machine which can support the entire EMBER2017 dataset.

H₆. The targeted adversarial attack can be prevented at test time.

From a visual inspection of the activation weights of test files against the average activation weights from the test data in the benign and malicious classes, it was clear that the Zero-day test files which succeeded in being misclassified as benign had an obvious pairwise separation from both the benign and malicious class. From analysing the activation weights at the pairwise level, it became apparent that for the majority of Zero-day files in Experiment 3 (180 out of 184), eight out of the ten neurons of the last hidden layer sat outside of one standard deviation from the benign class towards the malicious

class. While in the clean benign test data, an average of 6% contained the same activation weight pattern. After analysing the activation weights using a standard deviation derived from a larger (100,000) dataset, it became apparent that another method for calculating the deviation was required, as the larger dataset increased the standard deviation spread which prevented the original defence from functioning correctly. The Mean Absolute Deviation (MAD) was chosen, and an additional test was performed to identify if the activation weights existed outside of two MADs. The MAD defence was successful in identifying the majority of Zero-day files as suspicious for each of the adversarial poisoning attacks. The results are shown in Table 60, Table 61, and Table 62.

From the activation weight analysis, the evidence supports the hypothesis that it is possible to identify the targeted Zero-day file trying to bypass detection at test time. In a practical sense, a quarantine approach for files which meet the criteria of having suspicious activation weights should be implemented to mitigate the targeted adversarial poisoning attack.

6.5 Summary

In this chapter, a discussion of the adversarial attacks and defences was provided, along with an examination of the research questions and hypotheses and the results from their related experiments which either supported or rejected the hypothesis being tested. The adversarial attack section explored the results from the preliminary experiments, which provided the basis for how the targeted adversarial poisoning attacks were to be performed. The preliminary experiments identified that randomly selecting adversarial features performed just as well as manually selecting adversarial features, which benefited the targeted poisoning attack development for a black-box scenario. The targeted adversarial poisoning attack was examined with a focus on performing a compare and contrast the watermark poisoning attack which also used the EMBER dataset. The main difference between the two attacks is that the watermark attack performed feature analysis using SHAP values to identify which features should be modified for the attack to succeed, whereas the attack performed in this thesis did not perform any feature analysis, and instead randomly selected features from the imports section of the target Zero-day file to be used in the poisoning attack.

The adversarial defence section covered the defences developed to prevent targeted adversarial poisoning attacks by other researchers and the proposed MAD defence which was tested in this thesis. The previous defences which have been implemented to prevent targeted poisoning attacks from succeeding work by identifying outliers within the training data and removing them from the training set. Whereas the MAD defence identifies suspicious files at test time and quarantines the file for further analysis. The MAD defence works where it is assumed that the adversarial features used to poison the dataset blend in within the other clusters of benign features and cannot be easily removed through the use of anomaly analysis.

7 Conclusion

The final chapter concludes the work performed throughout this thesis. An overview of the research process and aims is provided, along with a summary of the contributions made throughout the research and, finally a discussion of future work which was considered but found to be out of scope for this thesis is stated.

7.1 Research Overview

This research had two aims. The first was to develop a targeted adversarial poisoning attack which would allow for Zero-day malware files to bypass detection. The second aim was to develop a defence which can mitigate the targeted adversarial poisoning attack. The research area was introduced in Chapter 1, which identified the uses of ML in the cyber security field and the associated known security issues in using ML. The domain of ML was examined for its contribution in defending against cyber security threats and general use applications. The significance of the field was quantified through an examination of the cyber security threat reports available. From identifying the significance of ML in the cyber security domain, two research questions were put forward to focus on one of the key areas of ML, cyber security.

A literature review was performed which covered three main areas, malware, machine learning, and adversarial machine learning. The malware section covered a variety of malware families and methods used for malware detection. At the beginning it was not known if the adversarial attack was going to concentrate on a ML model which could detect a particular malware family or a general malware detection application. A general malware detection application was chosen, and different malware files were used when experimenting on the adversarial poisoning attack. The machine learning section covered a variety of algorithms which are often used and the domains they are used in. The adversarial machine learning section covered the attacks and defences which have been developed, with a focus on the field of malware detection.

The research was completed in eight phases. The first two phases covered the acquisition and examination of datasets for the adversarial poisoning attacks and defences. The datasets acquired for use were the EMBER dataset, which belongs to a research framework for performing machine learning malware detection research, and from VirusShare, which contained the malware files used as the targets for the adversarial poisoning attacks. The feature engineering process of EMBER was evaluated to identify which area should be used as the target for the adversarial poisoning attacks. The imports feature section was chosen as it provided a large target feature space. No evaluation of the EMBER feature vectors was performed to identify which features contributed the most to the target model's classification. The poisoning attacks which were performed were done so under a strict

black-box scenario. Knowing the import section of the binary file would more than likely play a role in training the ML model was enough.

The third phase was the training of machine learning models. The EMBER framework provided a Gradient Boosted Decision Tree (GBDT) model and the required code to train additional GBDT models. The other type of model which was used for testing was a Multi-Layer Perceptron (MLP). The code for the MLP was adapted from another project that the EMBER creators had developed. The fourth, fifth, and sixth phases were all related to the testing of adversarial poisoning attacks. The fourth phase experimented with testing a general efficacy attack by injecting varying levels of adversarial features into the benign training data of the EMBER dataset. The general efficacy attacks were performed to identify if using the imports section as the basis of the targeted adversarial could be successful and if it were needed to manually select the features for poisoning. If the adversarial poisoning attacks in the general efficacy experiments failed to influence the trained model, then another set of features would have needed to be tested. The general efficacy experiments were successful in identifying the import section as an adequate attack vector.

The fifth and sixth phases involved the testing of the targeted adversarial poisoning attacks using both the GBDT and MLP models. Different percentages and methods of injection of adversarial features were performed in both the GBDT and MLP poisoning experiments. There were successful targeted adversarial poisoning attacks from both the GBDT and MLP experiments. MLP models were found to be more resilient than the GBDT models for targeted adversarial poisoning attacks. The eighth and ninth phases were about the defensive strategies to prevent the targeted adversarial poisoning attacks from succeeding. The MLP model was chosen as the target for the defence, as it is a more popular ML algorithm. The defensive technique developed is not model agnostic, it evaluates the activation weights of the neurons from the last hidden layer at test time for anomalies.

Current defences are performed on the training data to identify adversarial examples and remove them from the dataset. The identification of the adversarial examples requires a clean dataset to use for comparison, but it is not always possible to know if the dataset being used is in fact clean from adversarial poisoning, especially if the data is obtained online. The developed defence works under the assumption that the training dataset is poisoned since it is not possible to know for certain if it is not. The defence for the targeted adversarial poisoning attack examines the activation weights of the test file from the last hidden layer to see if there is an anomaly in a majority of the neurons. The anomaly being that the target file was classified as benign, but a majority of the activation weights were outside two MADs from the average benign activation weight.

Several hypotheses were developed and tested throughout this thesis to answer the proposed research questions. Their relationship with the proposed research questions and their results are shown in Table 63.

Table 63 - Research Question Relationships

Research Question	Related Hypotheses	Result
RQ1	H ₁ : A manual selection of features can be used for a successful adversarial poisoning attack	Accepted
	H ₂ : A random selection of features can be used for a successful adversarial poisoning attack	Accepted
	H ₃ : No more than 5% of the benign training data feature space is required to be poisoned to reduce the general efficacy of the model.	Rejected
	H ₄ : No more than 10% of the benign training data files are required to be poisoned to reduce the general efficacy of the model.	Rejected
	H ₅ : No more than 10% of the benign training data files are required to be poisoned for a targeted adversarial attack to succeed.	Accepted
RQ2	H ₆ : The targeted adversarial attack can be prevented at test time.	Accepted

7.2 Summary of Contributions

The aim of the research was twofold. First, to develop a targeted poisoning attack which allows for a certain Zero-day malware file to bypass detection, while not reducing the overall general efficacy of the malware detection model. Second, develop a defensive strategy to mitigate the adversarial attack. To achieve the end goal, a set of prerequisites were required to be completed first, which lead to the development of an adversarial attack and defence strategy.

The targeted adversarial poisoning attack was performed in a black-box scenario using a random selection of import library functions injected into different percentages of the benign training data. The import features were selected as the poisoning features after performing successful preliminary experiments which identified them as being a suitable feature set for reducing the efficacy of the target model. The targeted adversarial poisoning attack was performed in a black-box scenario, the

only information the attacker had of the ML architecture was that the model was trained to detect malicious PE32 files and that the import section of the PE32 files were used in the training of the model. No exploratory work was performed to identify what level of contribution the features of the PE32 file contributed to the models classification, unlike what was performed by Severi et al. (2020), where SHAP values were calculated to identify which features contributed the most to the model's classification.

The targeted adversarial attack was successful in allowing for previously detected Zero-day malware files to bypass detection by being misclassified as benign in both GBDT and MLP models. The results from the targeted adversarial poisoning attack have shown that it is possible to perform a targeted adversarial poisoning attack when the target feature area is comprised of a feature hash. In the watermark attack, Severi et al. (2020) avoided the vector dimensions of the EMBER training files which were a computed feature hash, which removed 2,316 of 2,351 of the input vectors. The authors only used values which were not computed into a feature hash as they needed to identify the contribution of the features before creating their watermark trigger. As feature hashing generates an irreversible hash digest, the authors would not have been able to identify which features contributed to the model's classification if they were to use any of the computed feature hash areas. The results from the experiments in this thesis show that it is possible to perform successful targeted adversarial attacks in a black-box scenario without computing any feature importance and the experiments show that targeted adversarial attacks can succeed if the input vectors are of a computed feature hash.

The first defensive strategy was to try and identify anomalous data at test time through the use of heatmaps. The intuition behind the defence was that as the target malicious Zero-day file was not modified in anyway, the majority of the file's features should lean towards malicious with the adversarial trigger features overwhelming the classification and misclassifying the file as benign. The idea was that the heatmap could be used to identify the features which contribute to the misclassification, and if there was an obvious anomaly within the feature importance (majority lean towards malicious but a small amount overwhelmingly pulls toward benign), the file would be quarantined for further examination. Unfortunately, the heatmap approach did not work visually for examining a one-dimensional plot.

The defence was modified from examining feature importance to evaluating the activation weights of the neurons from the penultimate layer of the model. The activation weights from the penultimate layer were examined as they contain all of the encoded data from the previous layers. The intuition behind the approach was that the activation weights of the target Zero-day file would differ significantly from the target benign activation weights from the test files, as the Zero-day file had not been modified and should lean more towards the activation weight values of the malicious class. It was found to be possible to identify on average ~80% of the target Zero-day files from the combined

targeted poisoning attacks by examining the activation weights of the neurons from the penultimate layer. The formula for detecting suspicious files was if six or more neurons were outside two MADs from the average activation weights of the test dataset, that file is suspicious and should be quarantined for further examination. A benefit of the MAD defence over other defences which remove poisoned data from the training set by identifying outliers is that the MAD defence does not require a clean dataset to generate the average activation weight values to perform the defence. The defence was shown to be successful when the average activation weights were generated on both clean and poisoned datasets.

7.3 Future Work

The proposed defensive technique which evaluated the activation weights from the last hidden layer is not able to identify targeted poisoning attacks when the attack succeeds with such a high proficiency to mask itself completely within the chosen target class. Future work for this research would concentrate on preventing the targeted adversarial attacks, which are successfully camouflaged within the target class and cannot be detected at test time by examining the activation weights.

Some additional experiments which were not performed as they were out of scope for this research include testing the training time defensive strategies utilised in the watermark attack paper on the targeted adversarial attacks performed in this research. It would be interesting to see if they can be detected at the training stage without removing a significant amount of genuine benign training data. Another experiment would be to see if the watermark attack can be detected at test time using the two MAD defence proposed in this thesis. Another defence for the watermark attack may be performed pre-emptively by hashing all the data vectors used for training. If the attacker is not able to distinguish which features contribute to the model's detection as the hash cannot be reversed, would it still be possible for the watermark attack to succeed?

The poisoning defence which has been integrated into the IBM Adversarial Robustness Toolbox did not function correctly when trying to identify poisoned samples in the MLP EMBER dataset. The poisoning defence has a set limit of 5,000 and the MLP EMBER dataset contains 50,000 samples. In the future, additional experiments which compare the efficacy of the poisoning defence in the IBM Toolbox and the MAD defence will be compared once a workaround for the 5,000 data limit has been developed.

Other areas of interest include investigating adversarial attacks and defences for ML software utilised in critical infrastructure and the defence industry. Defence and critical infrastructure are pivotal in for the well-being and safety of a nation. ML is being utilised in these areas in a variety of ways, from utilising ML security systems (e.g., malware detection, intrusion detection systems) to protect critical infrastructure or in the development of weapons, drones, and other military applications.

8 References

- Abdulkader, A., Lakshmiratan, A., & Zhang, J. (2016). Introducing DeepText: Facebook's text understanding engine. Retrieved from <http://code.facebook.com/posts/181565595577955/introducing-deeptext-facebook-s-text-understanding-engine/>
- Abhishek, B., & Goswami, R. T. (2017, 2017//). *DMDAM: Data Mining Based Detection of Android Malware*. Paper presented at the Proceedings of the First International Conference on Intelligent Computing and Communication, Singapore.
- Accenture. (2017). *Cost of Cyber Crime Study*. Retrieved from <https://www.accenture.com/us-en/insight-cost-of-cybercrime-2017>
- Accenture. (2019). *The Cost of Cyber Crime*. Retrieved from <https://www.accenture.com/acnmedia/pdf-96/accenture-2019-cost-of-cybercrime-study-final.pdf>
- Alam, S., Horspool, R. N., & Traore, I. (2014, 13-16 May 2014). *MARD: A Framework for Metamorphic Malware Analysis and Real-Time Detection*. Paper presented at the 2014 IEEE 28th International Conference on Advanced Information Networking and Applications.
- Allianz. (2020). Cyber attacks on critical infrastructure. Retrieved from <https://www.agcs.allianz.com/news-and-insights/expert-risk-articles/cyber-attacks-on-critical-infrastructure.html>
- Alpaydin, E. (2004). *Introduction to Machine Learning*. Cambridge, Mass: MIT Press.
- Anderson, H. S., Kharkar, A., Filar, B., & Roth, P. (2017). *Evading Machine Learning Malware Detection*. Paper presented at the Black Hat USA, Las Vegas, NV.
- Ani, U. P. D., He, H., & Tiwari, A. (2017). Review of cybersecurity issues in industrial critical infrastructure: manufacturing in perspective. *Journal of Cyber Security Technology*, 1(1), 32-74.
- Artificial neural network with layer coloring. (2013). Retrieved from https://en.wikipedia.org/wiki/File:Colored_neural_network.svg
- Banfield, R. E., Hall, L. O., Bowyer, K. W., & Kegelmeyer, W. P. (2007). A Comparison of Decision Tree Ensemble Creation Techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1), 173-180. doi:10.1109/TPAMI.2007.250609
- Banko, M., & Brill, E. (2002). Scaling to Very Very Large Corpora for Natural Language Disambiguation. *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. doi:10.3115/1073012.1073017
- Barreno, M., Nelson, B., Sears, R., Joseph, A. D., & Tygar, J. D. (2006). *Can machine learning be secure?* Paper presented at the ACM Symposium on Information, Computer and Communications Security, Taipei, Taiwan.
- Belinkov, Y., & Glass, J. (2019). Analysis Methods in Neural Language Processing: A Survey. *Transactions of the Association for Computational Linguistics*, 7, 49-72. doi:10.1162/tacl_a_00254
- Biggio, B., Fumera, G., Russu, P., Didaci, L., & Roli, F. (2015). Adversarial Biometric Recognition : A review on biometric system security from the adversarial machine-learning perspective. *IEEE Signal Processing Magazine*, 32(5), 31-41. doi:10.1109/MSP.2015.2426728
- Biggio, B., & Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84, 317-331.
- Bishop, C. (2006). *Pattern Recognition and Machine Learning* (1 ed.). New York: Springer.
- Brumley, D., Newsome, J., Song, D., Wang, H., & Jha, S. (2008). Theory and Techniques for Automatic Generation of Vulnerability-Based Signatures. *IEEE Transactions on Dependable and Secure Computing*, 5(4), 224-241. doi:10.1109/TDSC.2008.55

- Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2), 1153-1176. doi:10.1109/COMST.2015.2494502
- Cañedo, J., & Skjellum, A. (2016, 12-14 Dec. 2016). *Using machine learning to secure IoT systems*. Paper presented at the 2016 14th Annual Conference on Privacy, Security and Trust (PST).
- Cardenas, A., Amin, S., Sinopoli, B., Giani, A., Perrig, A., & Sastry, S. (2009). *Challenges for securing cyber physical systems*. Paper presented at the Workshop on future directions in cyber-physical systems security.
- Carlin, D., Cowan, A., O’Kane, P., & Sezer, S. (2017). The Effects of Traditional Anti-Virus Labels on Malware Detection Using Dynamic Runtime Opcodes. *IEEE Access*, 5, 17742-17752. doi:10.1109/ACCESS.2017.2749538
- Carlini, N., & Wagner, D. (2016). Defensive Distillation is Not Robust to Adversarial Examples. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1607.04311>
- Carlini, N., & Wagner, D. (2017). *Adversarial examples are not easily detected: Bypassing ten detection methods*. Paper presented at the Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security.
- Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., . . . Srivastava, B. (2018). Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. *ArXiv e-prints*. Retrieved from <https://arxiv.org/pdf/1811.03728.pdf>
- Chen, L., Ye, Y., & Bourlai, T. (2017, 11-13 Sept. 2017). *Adversarial Machine Learning in Malware Detection: Arms Race between Evasion Attack and Defense*. Paper presented at the 2017 European Intelligence and Security Informatics Conference (EISIC).
- Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1), 6. doi:10.1186/s12864-019-6413-7
- Chio, C., & Freeman, D. (2018). *Machine Learning and Security: Protecting Systems with Data and Algorithms*. Sebastopol, California: O’Reilly Media.
- Critical Infrastructure Centre. (2018). *Coverage of The Security of Critical Infrastructure Act 2018*. Retrieved from <https://cicentre.gov.au/document/P50S014>
- CrowdStrike. (2017). CrowdStrike Introduces Enhanced Endpoint Machine Learning Capabilities and Advanced Endpoint Protection Modules. Retrieved from <https://www.crowdstrike.com/resources/news/crowdstrike-introduces-enhanced-endpoint-machine-learning-capabilities-and-advanced-endpoint-protection-modules/>
- Cuckoo Sandbox. (2019). Retrieved from <https://cuckoosandbox.org/>
- Cybersecurity Ventures, & Herjavec Group. (2019). *2019 Official Annual Cybercrime Report*. Retrieved from <https://www.herjavecgroup.com/wp-content/uploads/2018/12/CV-HG-2019-Official-Annual-Cybercrime-Report.pdf>
- Cylance. CylancePROTECT: Artificial Intelligence Endpoint Security. Retrieved from https://cylance.com/en_us/products/our-products/enterprise-products/protect.html
- Deng, J., Dong, W., Socher, R., Li, L., Li, K., & Fei-Fei, L. (2009, 20-25 June 2009). *ImageNet: A large-scale hierarchical image database*. Paper presented at the 2009 IEEE Conference on Computer Vision and Pattern Recognition.
- Dietrich, C. J., Rossow, C., Freiling, F. C., Bos, H., Steen, M. V., & Pohlmann, N. (2011, 6-7 Sept. 2011). *On Botnets That Use DNS for Command and Control*. Paper presented at the 2011 Seventh European Conference on Computer Network Defense.
- Dunn, J. (2016). Introducing FBLeanner Flow: Facebook's AI backbone. Retrieved from <http://code.facebook.com/posts/1072626246134461/introducing-fbleanner-flow-facebook-s-ai-backbone/>
- Edgar, T. W., & Manz, D. O. (2017). *Research Methods for Cyber Security*: Syngress Publishing.
- ESET. (2020). Trojan Horse. Retrieved from <https://www.eset.com/uk/types-of-cyber-threats/trojan-horse/>

- Feroze, M., Baig, Z., & Johnstone, M. (2015). *A Two-Tiered User Feedback-based Approach for Spam Detection*. Paper presented at the Tenth International Conference on Systems and Networks Communications, Barcelona, Spain.
- Fredrikson, M., Jha, S., & Ristenpart, T. (2015). *Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures*. Paper presented at the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA.
- Friedman, J., Hastie, T., & Tibshirani, R. (2017). *The Elements of Statistical Learning*. New York: Springer.
- Galliers, R. D. (1990). *Choosing Appropriate Information Systems Research Approaches: A Revised Taxonomy*. Paper presented at the The Information Systems Research Arena of the 90's: Challenges, Perceptions and Alternative Approaches - Proc. IFIP TC8 WG8.2 Conference. , Copenhagen, Denmark.
- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). Sebastopol, California: O'Reilly Media, Inc.
- Gibert, D., Mateu, C., & Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, 153, 102526.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). *Generative Adversarial Networks*. Paper presented at the Advances in Neural Information Processing Systems, Montreal, Canada.
- Goodfellow, I., Shlens, J., & Szegedy, C. (2015). *Explaining and Harnessing Adversarial Examples*. Paper presented at the ICLR, San Diego, CA, USA.
- Google. (2018). Google I/O'18: Google Keynote Retrieved from <https://www.youtube.com/watch?v=ogfYd705cRs>
- Gretton, A., Borgwardt, K., Rasch, M., Schölkopf, B., & Smola, A. (2012). A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1), 723-773.
- Griggs, T., & Wakabayashi, D. (2018). How a Self-Driving Uber Killed a Pedestrian in Arizona. Retrieved from <https://www.nytimes.com/interactive/2018/03/20/us/self-driving-uber-pedestrian-killed.html#>
- Grosse, K., Manoharan, P., Papernot, N., Backes, M., & McDaniel, P. (2017). On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 13. Retrieved from <https://arxiv.org/abs/1702.06280>
- Grosse, K., Papernot, N., Manoharan, P., Backes, M., & McDaniel, P. (2016). Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1606.04435>
- Gu, T., Liu, K., Dolan-Gavitt, B., & Garg, S. (2019). BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access*, 7, 47230-47244. doi:10.1109/ACCESS.2019.2909068
- Guba, E. G. (1981). Criteria for assessing the trustworthiness of naturalistic inquiries. *ECTJ*, 29(2), 75. doi:10.1007/BF02766777
- Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., . . . Webster, D. R. (2016). Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA*, 316(22), 2402-2410. doi:10.1001/jama.2016.17216
- Guo, F., Ferrie, P., & Chiueh, T. (2008, 2008//). *A Study of the Packer Problem and Its Solutions*. Paper presented at the Recent Advances in Intrusion Detection, Berlin, Heidelberg.
- Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, 187, 27-48. doi:<https://doi.org/10.1016/j.neucom.2015.09.116>
- Hackeling, G. (2017). *Mastering Machine Learning with scikit-learn*: Packt Publishing Ltd.
- Haykin, S. (2008). *Neural Networks and Learning Machines* (Third ed.). Upper Saddle River, New Jersey: Pearson.

- Hinton, G., Vinyals, O., & Dean, J. (2014). *Distilling the Knowledge in a Neural Network*. Paper presented at the NIPS Deep Learning Workshop, Montréal, Canada
- Hu, W., & Tan, Y. (2017). Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1702.05983>
- Huang, L., Joseph, A. D., Nelson, B., Rubinstein, B., & Tygar, J. D. (2011). *Adversarial Machine Learning*. Paper presented at the 4th ACM workshop on Security and Artificial Intelligence, Chicago, Illinois, USA.
- IBM. (2016). IBM Study: Businesses More likely to Pay Ransomware than Consumers [Press release]. Retrieved from <http://www-03.ibm.com/press/us/en/pressrelease/51230.wss>
- Jagielski, M., Severi, G., Hager, N., & Oprea, A. (2020). Subpopulation Data Poisoning Attacks. *ArXiv e-prints*. Retrieved from <https://arxiv.org/pdf/2006.14026>
- Janai, J., Güney, F., Behl, A., & Geiger, A. (2020). Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3), 1-308. doi:10.1561/06000000079
- Jianliang, M., Haikun, S., & Ling, B. (2009, 15-17 May 2009). *The Application on Intrusion Detection Based on K-means Cluster Algorithm*. Paper presented at the 2009 International Forum on Information Technology and Applications.
- Juniper. (2017). *Cybercrime & the Internet of Threats 2017*. Retrieved from <https://www.juniperresearch.com/document-library/white-papers/cybercrime-the-internet-of-threats-2017>
- Kan, M. (2018). Hackers Invade YouTube Ads To Mine Cryptocurrency. Retrieved from <https://au.pcmag.com/security/51512/hackers-invade-youtube-ads-to-mine-cryptocurrency>
- Lee, P. (2016). Learning from Tay's introduction. Retrieved from <http://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/#sm.0001gomcmq6ebdchzas1zmggkqlnq>
- Liu, Q., Li, P., Zhao, W., Cai, W., Yu, S., & Leung, M. (2018). A Survey on Security Threats and Defensive Techniques of Machine Learning: A Data Driven View. *IEEE Access*, PP(99), 1-1. doi:10.1109/ACCESS.2018.2805680
- Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., & Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234, 11-26. doi:<https://doi.org/10.1016/j.neucom.2016.12.038>
- Malwarebytes. (2020). *State of Malware*. Retrieved from https://resources.malwarebytes.com/files/2020/02/2020_State-of-Malware-Report.pdf
- Marr, B. (2018). Machine Learning In Practice: How Does Amazon's Alexa Really Work? Retrieved from <https://www.forbes.com/sites/bernardmarr/2018/10/05/how-does-amazons-alexa-really-work/#369f2d291937>
- Martin, A., Menéndez, H. D., & Camacho, D. (2016, 24-29 July 2016). *Genetic boosting classification for malware detection*. Paper presented at the 2016 IEEE Congress on Evolutionary Computation (CEC).
- McAfee. (2017). *The Good, the Bad, and the Unknown*. Retrieved from http://www.techdata.com/mcafee/files/MCAFEE_wp_appcontrol-good-bad-unknown.pdf
- Merabti, M., Kennedy, M., & Hurst, W. (2011, 29-31 March 2011). *Critical infrastructure protection: A 21st century challenge*. Paper presented at the 2011 International Conference on Communications and Information Technology (ICCIT).
- Nash, C. (2016). Microsoft's AI-Controlled Twitter Account Turned into a Digital Nazi. Retrieved from <http://www.breitbart.com/tech/2016/03/24/meet-tay-ai-twitter-trolls-turned-into-nazi/>
- Nelson, B., Barreno, M., Chi, F. J., Joseph, A. D., Rubinstein, B. I. P., Saini, U., . . . Xia, K. (2008). *Exploiting Machine Learning to Subvert Your Spam Filter*. Paper presented at the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, San Francisco, California.
- Neuman, W. L. (2014). *Social research methods: Qualitative and quantitative approaches* (7th ed.). Harlow, Essex: Pearson New International.

- Newsome, J., Karp, B., & Song, D. (2006). *Paragraph: Thwarting Signature Learning by Training Maliciously*. Paper presented at the Recent Advances in Intrusion Detection, Berlin, Heidelberg.
- NTSB. (2019). *Preliminary Report Highway: HWY18MH010*. Retrieved from <https://www.nts.gov/investigations/AccidentReports/Pages/HWY18MH010-prelim.aspx>
- O’Kane, P., Sezer, S., & McLaughlin, K. (2011). Obfuscation: The Hidden Malware. *IEEE Security & Privacy*, 9(5), 41-47. doi:10.1109/MSP.2011.98
- Obulesu, O., Mahendra, M., & ThirlokReddy, M. (2018, 11-12 July 2018). *Machine Learning Techniques and Tools: A Survey*. Paper presented at the 2018 International Conference on Inventive Research in Computing Applications (ICIRCA).
- Oliver, J. J., Hou, C., Dia, L., Liang, Y., & Rihn, J. (2013). US Patent No.
- Papernot, N., & McDaniel, P. (2017). Extending Defensive Distillation. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1705.05264>
- Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., & Swami, A. (2017). *Practical Black-Box Attacks against Machine Learning*. Paper presented at the Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates.
- Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. (2016). *Distillation As A Defense to Adversarial Perturbations Against Deep Neural Networks*. Paper presented at the IEEE Symposium on Security and Privacy, San Jose, CA, USA.
- Paudice, A., Muñoz-González, L., Gyorgy, A., & Lupu, E. (2018). Detection of Adversarial Training Examples in Poisoning Attacks through Anomaly Detection. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1802.03041>
- Paudice, A., Muñoz-González, L., & Lupu, E. (2018). Label Sanitization against Label Flipping Poisoning Attacks. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1803.00992>
- Peacock, M. (2019). *Anomaly Detection in BACnet/IP managed Building Automation Systems*. (PhD). Edith Cowan University, Retrieved from <https://ro.ecu.edu.au/theses/2178>
- Perols, J. (2011). Financial Statement Fraud Detection: An Analysis of Statistical and Machine Learning Algorithms. *AUDITING: A Journal of Practice & Theory*, 30(2), 19-50. doi:10.2308/ajpt-50009
- Pitropakis, N., Panaousis, E., Giannetsos, T., Anastasiadis, E., & Loukas, G. (2019). A taxonomy and survey of attacks against machine learning. *Computer Science Review*, 34, 100199. doi:<https://doi.org/10.1016/j.cosrev.2019.100199>
- Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., . . . Nicholas, C. (2018). An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, 14(1), 1-20. doi:10.1007/s11416-016-0283-1
- Raj, S. B. E., & Portia, A. A. (2011). *Analysis on credit card fraud detection methods*. Paper presented at the 2011 International Conference on Computer, Communication and Electrical Technology (ICCCET).
- Rajab Abu, M., Ballard, L., Lutz, N., Mavrommatis, P., & Provos, N. (2013). *CAMP: Content-Agnostic Malware Protection*. Paper presented at the NDSS, San Diego, CA.
- Randazzo, R. (2020a). Operator of self-driving Uber charged with negligent homicide in 2018 fatal crash. *azcentral*. Retrieved from <https://www.azcentral.com/story/money/business/consumers/2020/09/15/rafaela-vasquez-charged-negligent-homicide-2018-uber-crash-arizona/5810172002/>
- Randazzo, R. (2020b). Who was really at fault in fatal Uber crash? Here's the whole story. *azcentral*. Retrieved from <https://www.azcentral.com/story/news/local/tempe/2019/03/17/one-year-after-self-driving-uber-rafaela-vasquez-behind-wheel-crash-death-elaine-herzberg-tempe/1296676002/>
- Rebala, G., Ravi, A., & Churiwala, S. (2019). *An Introduction to Machine Learning*. New York: Springer International Publishing.

- Reuters. (2017). Cyber attack hits 200,000 in at least 150 countries: Europol. *Reuters*. Retrieved from <https://www.reuters.com/article/us-cyber-attack-europol-idUSKCN18A0FX>
- Rewari, M. (2020). What's new with Cortana, your personal productivity assistant. Retrieved from <https://techcommunity.microsoft.com/t5/microsoft-365-blog/what-s-new-with-cortana-your-personal-productivity-assistant/ba-p/1675341>
- Rogers, S., & Girolami, M. (2016). *A First Course in Machine Learning, Second Edition*: Chapman \& Hall/CRC.
- Rosenberg, I., Shabtai, A., Elovici, Y., & Rokach, L. (2018). Low Resource Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1804.08778>
- Rosenberg, I., Shabtai, A., Rokach, L., & Elovici, Y. (2018). Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1707.05970>
- Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3), 210-229. doi:10.1147/rd.33.0210
- Saxe, J., & Berlin, K. (2015). *Deep neural network based malware detection using two dimensional binary program features*. Paper presented at the Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). <https://doi.org/10.1109/MALWARE.2015.7413680>
- Scharre, P. (2019). *Army of None: Autonomous Weapons and the Future of War*. New York, United States: W. W. Norton Company.
- Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2001, 14-16 May 2000). *Data mining methods for detection of new malicious executables*. Paper presented at the Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001.
- Scikit. Precision-Recall Curve. In.
- Scikit. (2014). Underfitting vs. Overfitting. In.
- Severi, G., Meyer, J., Coull, S., & Oprea, A. (2020). Exploring Backdoor Poisoning Attacks Against Malware Classifiers. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/2003.01031>
- Shafiq, M. Z., Tabish, S. M., Mirza, F., & Farooq, M. (2009, 2009/). *PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime*. Paper presented at the Recent Advances in Intrusion Detection, Berlin, Heidelberg.
- Sharif, M., Bhagavatula, S., Bauer, L., & Reiter, M. K. (2016). *Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition*. Paper presented at the ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria.
- Shen, S., Jiang, H., & Zhang, T. (2012). Stock market forecasting using machine learning algorithms. *Department of Electrical Engineering, Stanford University, Stanford, CA*, 1-5.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., . . . Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 484. doi:10.1038/nature16961
- Simonite, T. (2018). When It Comes to Gorillas, Google Photos Remains Blind. *Wired*. Retrieved from <https://www.wired.com/story/when-it-comes-to-gorillas-google-photos-remains-blind/>
- Siri Team. (2017). Hey Siri: An On-device DNN-powered Voice Trigger for Apple's Personal Assistant. Retrieved from <https://machinelearning.apple.com/research/hey-siri>
- Sophos. (2017). Sophos Adds Advanced Machine Learning to Its Next-Generation Endpoint Protection Portfolio with Acquisition of Invincea. Retrieved from <https://www.sophos.com/en-us/press-office/press-releases/2017/02/sophos-adds-advanced-machine-learning-to-its-next-generation-endpoint-protection-portfolio.aspx>
- Suciu, O., Marginean, R., Kaya, Y., Daume, H., & Dumitras, T. (2018). When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1803.06975>
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*: MIT Press.

- Symantec. (2018). How does Symantec Endpoint Protection use advanced machine learning? Retrieved from https://support.symantec.com/en_US/article.HOWTO125816.html
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2014). *Intriguing properties of neural networks*. Paper presented at the International Conference on Learning Representations, Banff, Canada.
- Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). *Deepface: Closing the Gap to Human-Level Performance in Face Verification*. Paper presented at the Conference on Computer Vision and Pattern Recognition (CVPR) Columbus, OH.
- Tesla. (2018). An Update on Last Week's Accident. Retrieved from <https://www.tesla.com/blog/update-last-week%E2%80%99s-accident>
- Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). *DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars*. Paper presented at the International Conference on Software Engineering, Gothenburg, Sweden.
- Trafalis, T. B., & Ince, H. (2000, 2000). *Support vector machine for regression and applications to financial forecasting*. Paper presented at the Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium.
- Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. (2018). *Ensemble Adversarial Training: Attacks and Defenses*. Paper presented at the ICLR, Vancouver, Canada.
- Tran, B., Li, J., & Madry, A. (2018). *Spectral signatures in backdoor attacks*. Paper presented at the International Conference on Neural Information Processing Systems, Montreal, Canada.
- Ucci, D., Aniello, L., & Baldoni, R. (2017). Survey on the Usage of Machine Learning Techniques for Malware Analysis. *ACM Trans. Web*, 1(1), 34.
- Vogel, D. R., & Wetherbe, J. C. (1984). MIS Research: A Profile of Leading Journals and Universities". *Data Base*, 16(3), 3-14.
- Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., & Zhao, B. Y. (2019, 19-23 May 2019). *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*. Paper presented at the 2019 IEEE Symposium on Security and Privacy (SP).
- Williamson, K., & Johanson, G. (2017). *Research methods: Information, systems, and contexts*: Chandos Publishing.
- Wittel, G. L., & Wu, S. F. (2004). *On Attacking Statistical Spam Filters*. Paper presented at the First Conference on Email and Anti-Spam, Mountain View, CA, USA.
- Wong, E., Rice, L., & Kolter, J. Z. (2020). Fast is better than free: Revisiting adversarial training. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/2001.03994>
- Wood, A., & Johnstone, M. (2020). *Detection of False Negatives in Adversarial Machine Learning*. Paper presented at the To appear in: Cyber Defence Next Generation Technology & Science Conference, Brisbane.
- Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012, 9-10 Aug. 2012). *DroidMat: Android Malware Detection through Manifest and API Calls Tracing*. Paper presented at the 2012 Seventh Asia Joint Conference on Information Security.
- Wu, W.-C., & Hung, S.-H. (2014). *DroidDolphin: a dynamic Android malware detection framework using big data and machine learning*. Paper presented at the Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, Towson, Maryland. <https://doi.org/10.1145/2663761.2664223>
- Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., . . . Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv e-prints*. Retrieved from <https://arxiv.org/abs/1609.08144>
- Wyke, J. (2012). ZeroAccess. Retrieved from UK: <https://news-sophos.go-vip.net/wp-content/uploads/sites/2/2012/04/zeroaccess2.pdf>

- Xiao, F., Lin, Z., Sun, Y., & Ma, Y. (2019). Malware Detection Based on Deep Learning of Behavior Graphs. *Mathematical Problems in Engineering*, 2019, 8195395. doi:10.1155/2019/8195395
- Xiao, H., Biggio, B., Brown, G., Fumera, G., Eckert, C., & Roli, F. (2015). *Is Feature Selection Secure against Training Data Poisoning?* Paper presented at the ICML, Lille, France.
- Yang, W., Wang, S., & Johnstone, M. (2020). *A Comparative Study of ML-ELM and DNN for Intrusion Detection*.
- Youn, S., & McLeod, D. (2007). *A Comparative Study for Email Classification*. Paper presented at the Advances and Innovations in Systems, Computing Sciences and Software Engineering, Dordrecht.
- Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent Trends in Deep Learning Based Natural Language Processing [Review Article]. *IEEE Computational Intelligence Magazine*, 13(3), 55-75. doi:10.1109/MCI.2018.2840738
- Zeadally, S., Adi, E., Baig, Z., & Khan, I. A. (2020). Harnessing Artificial Intelligence Capabilities to Improve Cybersecurity. *IEEE Access*, 8, 23817-23837. doi:10.1109/ACCESS.2020.2968045
- Zukauskas, P., Vveinhardt, J., & Andriukaitienė, R. (2018). Philosophy and Paradigm of Scientific Research. In.

9 Appendix A – Machine Learning Model Code

EMBER GBDT Code

```
def optimize_model(data_dir):
    """
    Run a grid search to find the best LightGBM parameters
    """
    # Read data
    X_train, y_train = read_vectorized_features(data_dir, subset="train")

    # Filter unlabeled data
    train_rows = (y_train != -1)

    # read training dataset
    X_train = X_train[train_rows]
    y_train = y_train[train_rows]

    # score by roc auc
    # we're interested in low FPR rates, so we'll consider only the AUC for FPRs in [0,5e-3]
    score = make_scorer(roc_auc_score, max_fpr=5e-3)

    # define search grid
    param_grid = {
        'boosting_type': ['gbdt'],
        'objective': ['binary'],
        'num_iterations': [500, 1000],
        'learning_rate': [0.005, 0.05],
        'num_leaves': [512, 1024, 2048],
        'feature_fraction': [0.5, 0.8, 1.0],
        'bagging_fraction': [0.5, 0.8, 1.0]
    }
    model = lgb.LGBMClassifier(boosting_type="gbdt", n_jobs=-1, silent=True)

    # each row in X_train appears in chronological order of "appeared"
    # so this works for progressive time series splitting
    progressive_cv = TimeSeriesSplit(n_splits=3).split(X_train)

    grid = GridSearchCV(estimator=model, cv=progressive_cv, param_grid=param_grid,
                        scoring=score, n_jobs=1, verbose=3)
    grid.fit(X_train, y_train)

    return grid.best_params_


def train_model(data_dir, params={}):
    """
    Train the LightGBM model from the EMBER dataset from the vectorized features
    """
    # update params
    params.update({"application": "binary"})

    # Read data
    X_train, y_train = read_vectorized_features(data_dir, "train")
```

```
# Filter unlabeled data
train_rows = (y_train != -1)

# Train
lgbm_dataset = lgb.Dataset(X_train[train_rows], y_train[train_rows])
lgbm_model = lgb.train(params, lgbm_dataset)
#lgbm_model = lgb.fit(params, lgbm_dataset)

return lgbm_model
```

EMBER MLP Code

```
def train_model(X_train, X_test, y_train, y_test, virus_sha256):
    model = simple_multilayer.create_model(
        input_shape=(X_train.shape[1], ),
        input_dropout=0.05,
        hidden_dropout=0.1,
        hidden_layers=[2048, 1024, 512, 256, 10]
    )
    model.fit(X_train, y_train,
        batch_size=128,
        epochs=100,
        verbose=1,
        callbacks=[
            EarlyStopping( patience=20 ),
            ModelCheckpoint( '/media/me/My Passport/Models/'+virus_sha256+'_multilayer.h5',
save_best_only=True),
            ReduceLROnPlateau( patience=5, verbose=1)],
        validation_data=(X_test, y_test))

    model = load_model('/media/me/My Passport/Models/'+virus_sha256+'_multilayer.h5')

    y_pred = model.predict(X_test)
    roc = roc_auc_score(y_test, y_pred)
    a,b,c,d,e,f = common.summarize_performance(y_pred, y_test, "Multilayer perceptron")

    return b,e,f,model,roc
```

10 Appendix B – Experiment Result Tables

Manually Selected Functions

“user32.dll” Import Functions

Five Functions – user32.dll

‘SetRect’, ‘GetSubMenu’, ‘GetDCEX’, ‘GetMessageTime’, ‘SetWindowRgn’

Training file	Unique Count	Library	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	12,824		654,156	54,533	8.336%
train_features_1.JSONI	11,664		590,103	49,572	8.4%
train_features_2.JSONI	13,655		694,225	57,636	8.302%
train_features_3.JSONI	16,678		835,414	70,558	8.445%
train_features_4.JSONI	17,940		1,313,214	64,583	4.917%
train_features_5.JSONI	11,772		511,952	51,396	10.039%
Total	84,533		4,599,064	348,278	7.572%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_user32_5_5	0.991	1.764	0.537	99.90235814	98.236
manual_user32_5_10	0.999	1.952	0.55	99.90298298	98.048
manual_user32_5_15	0.997	1.883	0.543	99.90653725	98.117
manual_user32_5_20	0.995	1.914	0.557	99.90367065	98.086
manual_user32_5_25	0.997	1.946	0.558	99.89786455	98.054
manual_user32_5_30	1	1.961	0.578	99.90072732	98.039
manual_user32_5_35	1	2.155	0.6	99.89211274	97.845
manual_user32_5_40	0.999	1.981	0.596	99.89589024	98.019
manual_user32_5_45	1	2.152	0.622	99.8941914	97.848
manual_user32_5_50	0.997	2.111	0.628	99.8889534	97.889
manual_user32_5_55	0.999	2.182	0.64	99.88439924	97.818
manual_user32_5_60	1	2.319	0.663	99.88538346	97.681
manual_user32_5_65	0.998	2.347	0.692	99.88158333	97.653
manual_user32_5_70	1	2.395	0.705	99.88058659	97.605
manual_user32_5_75	0.991	2.559	0.749	99.86732209	97.441
manual_user32_5_80	0.998	2.698	0.769	99.86417246	97.302
manual_user32_5_85	0.998	3.28	0.833	99.83772575	96.72
manual_user32_5_90	0.995	3.612	0.874	99.81581328	96.388
manual_user32_5_95	0.991	5.376	0.931	99.75475833	94.624
manual_user32_5_100	0.571	58.177	0.999	97.7141671	41.823

Ten Functions – user32.dll

'SetRect', 'GetSubMenu', 'GetDCEX', 'GetMessageTime', 'SetWindowRgn', 'SetScrollPos',
'IntersectRect', 'CallNextHookEx', 'GetDlgItem', 'ReleaseDC'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	12,824	654,156	101,528	15.52%
train_features_1.JSONI	11,664	590,103	92,857	15.735%
train_features_2.JSONI	13,655	694,225	108,299	15.599%
train_features_3.JSONI	16,678	835,414	131,789	15.775%
train_features_4.JSONI	17,940	1,313,214	117,992	8.984%
train_features_5.JSONI	11,772	511,952	97,274	19%
Total	84,533	4,599,064	649,739	14.127%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_user32_10_5	0.999	1.695	0.531	99.91032533	98.305
manual_user32_10_10	0.988	1.813	0.542	99.90556271	98.187
manual_user32_10_15	1	1.803	0.544	99.90885283	98.197
manual_user32_10_20	0.999	1.856	0.555	99.90716998	98.144
manual_user32_10_25	0.998	1.909	0.572	99.89671515	98.091
manual_user32_10_30	0.999	1.944	0.579	99.89477113	98.056
manual_user32_10_35	0.997	1.958	0.582	99.90150219	98.042
manual_user32_10_40	1	1.946	0.599	99.89737914	98.054
manual_user32_10_45	0.999	2.076	0.624	99.88338406	97.924
manual_user32_10_50	0.997	2.072	0.634	99.88944858	97.928
manual_user32_10_55	0.996	2.014	0.65	99.89016454	97.986
manual_user32_10_60	0.996	2.123	0.672	99.88576935	97.877
manual_user32_10_65	1	2.349	0.703	99.8737667	97.651
manual_user32_10_70	0.992	2.21	0.704	99.87626327	97.79
manual_user32_10_75	0.993	2.452	0.75	99.86222377	97.548
manual_user32_10_80	0.998	2.684	0.783	99.85117525	97.316
manual_user32_10_85	0.997	2.93	0.832	99.84456597	97.07
manual_user32_10_90	0.992	3.439	0.877	99.81800112	96.561
manual_user32_10_95	0.992	5.349	0.935	99.7390699	94.651
manual_user32_10_100	0	100	1	97.49084004	0

“shell32.dll” Import Functions

Five Functions – shell32.dll

‘ShellExecuteA’, ‘SHGetFolderPathA’, ‘ShellExecuteExA’, ‘SHFileOperationA’,
‘ShellMessageBoxW’

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	7,187	33,299	31,351	94.149%
train_features_1.JSONI	6,304	28,970	27,989	96.613%
train_features_2.JSONI	7,447	34,450	32,645	94.760%
train_features_3.JSONI	9,537	40,372	41,885	103.747%
train_features_4.JSONI	11,584	78,278	52,666	67.280%
train_features_5.JSONI	5,691	26,453	25,257	95.478%
Total	47,750	241,822	211,793	87.582%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_shell32_5_5	1	1.791	0.526	99.90990048	98.209
manual_shell32_5_10	0.998	1.795	0.539	99.90831316	98.205
manual_shell32_5_15	1	1.789	0.537	99.91043297	98.211
manual_shell32_5_20	1	1.864	0.544	99.9071337	98.136
manual_shell32_5_25	0.997	1.917	0.56	99.90339206	98.083
manual_shell32_5_30	0.994	1.959	0.56	99.89858747	98.041
manual_shell32_5_35	0.998	2.001	0.571	99.90563369	97.999
manual_shell32_5_40	0.997	1.873	0.576	99.90451775	98.127
manual_shell32_5_45	1	2.031	0.596	99.90170271	97.969
manual_shell32_5_50	1	2.059	0.6	99.89913858	97.941
manual_shell32_5_55	0.996	2.138	0.622	99.89766369	97.862
manual_shell32_5_60	0.999	2.106	0.628	99.89214973	97.894
manual_shell32_5_65	0.998	2.391	0.657	99.88851385	97.609
manual_shell32_5_70	0.996	2.288	0.665	99.88793105	97.712
manual_shell32_5_75	0.999	2.431	0.683	99.88038944	97.569
manual_shell32_5_80	0.994	2.974	0.728	99.85943153	97.026
manual_shell32_5_85	0.997	2.919	0.766	99.85234237	97.081
manual_shell32_5_90	0.993	3.661	0.818	99.82451533	96.339
manual_shell32_5_95	0.991	5.045	0.895	99.75306615	94.955
manual_shell32_5_100	0.965	34.63	0.992	98.95937595	65.37

Ten Functions – shell32.dll

'ShellExecuteA', 'SHGetFolderPathA', 'ShellExecuteExA', 'SHFileOperationA',
 'ShellMessageBoxW', 'FindExecutableA', 'ShellAboutA', 'SHGetDataFromIDListA',
 'SHGetDiskFreeSpaceA', 'SHCreateShellItem'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	7,187	33,299	66,988	201.171%
train_features_1.JSONI	6,304	28,970	59,308	204.722%
train_features_2.JSONI	7,447	34,450	69,552	201.892%
train_features_3.JSONI	9,537	40,372	89,093	220.680%
train_features_4.JSONI	11,584	78,278	110,061	140.602%
train_features_5.JSONI	5,691	26,453	53,497	202.234%
Total	47,750	241,822	448,499	185.466%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_shell32_10_5	1	1.914	0.528	99.90698587	98.086
manual_shell32_10_10	0.998	1.938	0.543	99.90482298	98.062
manual_shell32_10_15	1	1.867	0.545	99.90235401	98.133
manual_shell32_10_20	0.999	1.982	0.553	99.90067669	98.018
manual_shell32_10_25	0.999	1.91	0.566	99.9027826	98.09
manual_shell32_10_30	0.999	1.924	0.576	99.89568811	98.076
manual_shell32_10_35	0.994	2.174	0.594	99.89542627	97.826
manual_shell32_10_40	0.999	2.016	0.586	99.89464113	97.984
manual_shell32_10_45	0.998	2.064	0.605	99.89176835	97.936
manual_shell32_10_50	0.995	2.18	0.612	99.89388337	97.82
manual_shell32_10_55	1	2.238	0.627	99.88926653	97.762
manual_shell32_10_60	1	2.351	0.647	99.88536494	97.649
manual_shell32_10_65	1	2.325	0.666	99.88691268	97.675
manual_shell32_10_70	0.999	2.46	0.683	99.88018014	97.54
manual_shell32_10_75	0.997	2.512	0.712	99.87590608	97.488
manual_shell32_10_80	0.999	2.759	0.741	99.86618962	97.241
manual_shell32_10_85	0.99	3.164	0.78	99.8480135	96.836
manual_shell32_10_90	0.997	3.619	0.833	99.82736403	96.381
manual_shell32_10_95	0.991	6.251	0.917	99.73005722	93.749
manual_shell32_10_100	0.81	53.969	0.997	98.30671245	46.031

“advapi32.dll” Import Functions

Five Functions – advapi32.dll

‘GetUserNameA’, ‘CloseServiceHandle’, ‘CryptHashData’, ‘RegDeleteKeyA’, ‘RegFlushKey’

Training file	Unique Count	Library	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	12,576		159,639	57,040	35.73%
train_features_1.JSONI	15,956		237,016	74,859	31.583%
train_features_2.JSONI	16,272		227,814	75,675	33.217%
train_features_3.JSONI	14,039		159,156	62,642	39.358%
train_features_4.JSONI	15,801		217,846	72,561	33.308%
train_features_5.JSONI	9,910		116,957	45,195	38.642%
Total	84,554		1,118,428	387,972	34.689%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_advapi32_5_5	0.997	1.842	0.538	99.90636527	98.158
manual_advapi32_5_10	1	1.819	0.537	99.9080867	98.181
manual_advapi32_5_15	1	1.87	0.56	99.90581686	98.13
manual_advapi32_5_20	0.999	1.867	0.552	99.91025671	98.133
manual_advapi32_5_25	1	1.969	0.574	99.89481628	98.031
manual_advapi32_5_30	0.994	1.9	0.571	99.90211657	98.1
manual_advapi32_5_35	0.999	1.871	0.578	99.90620146	98.129
manual_advapi32_5_40	0.995	1.791	0.583	99.90244471	98.209
manual_advapi32_5_45	0.999	2.008	0.601	99.90469535	97.992
manual_advapi32_5_50	0.995	1.959	0.612	99.90514259	98.041
manual_advapi32_5_55	0.997	2.047	0.638	99.89600354	97.953
manual_advapi32_5_60	0.996	2.203	0.659	99.89161551	97.797
manual_advapi32_5_65	0.996	2.137	0.661	99.89306434	97.863
manual_advapi32_5_70	0.997	2.3	0.689	99.88887363	97.7
manual_advapi32_5_75	0.999	2.463	0.727	99.88226514	97.537
manual_advapi32_5_80	1	2.648	0.762	99.87081377	97.352
manual_advapi32_5_85	0.994	2.898	0.792	99.86369187	97.102
manual_advapi32_5_90	0.998	3.404	0.853	99.83717186	96.596
manual_advapi32_5_95	0.989	5.364	0.918	99.77490468	94.636
manual_advapi32_5_100	0.419	56.358	0.998	98.14491437	43.642

Ten Functions – advapi32.dll

'GetUserNameA', 'CloseServiceHandle', 'CryptHashData', 'RegDeleteKeyA', 'RegFlushKey',
'StartServiceA', 'ControlService', 'AddAce', 'RegConnectRegistryW', 'CreateProcessAsUserW'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	12,576	159,639	118,403	74.169%
train_features_1.JSONI	15,956	237,016	153,272	64.667%
train_features_2.JSONI	16,272	227,814	155,620	68.310%
train_features_3.JSONI	14,039	159,156	131,162	82.410%
train_features_4.JSONI	15,801	217,846	150,030	68.869%
train_features_5.JSONI	9,910	116,957	93,456	79.906%
Total	84,554	1,118,428	801,943	71.702%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_advapi32_10_5	0.996	1.797	0.534	99.90283649	98.203
manual_advapi32_10_10	1	1.773	0.551	99.90875651	98.227
manual_advapi32_10_15	0.999	1.813	0.555	99.90870546	98.187
manual_advapi32_10_20	0.997	1.928	0.555	99.90637989	98.072
manual_advapi32_10_25	0.996	1.841	0.569	99.91028406	98.159
manual_advapi32_10_30	0.998	1.951	0.586	99.9029045	98.049
manual_advapi32_10_35	0.999	1.935	0.585	99.89508652	98.065
manual_advapi32_10_40	0.994	2.08	0.594	99.90153695	97.92
manual_advapi32_10_45	0.998	1.945	0.606	99.89935235	98.055
manual_advapi32_10_50	0.993	2.118	0.618	99.89860348	97.882
manual_advapi32_10_55	0.999	1.982	0.643	99.89675165	98.018
manual_advapi32_10_60	0.997	2.301	0.663	99.89119558	97.699
manual_advapi32_10_65	0.996	2.456	0.678	99.88695771	97.544
manual_advapi32_10_70	1	2.51	0.706	99.88131724	97.49
manual_advapi32_10_75	1	2.801	0.74	99.87840738	97.199
manual_advapi32_10_80	0.997	2.756	0.764	99.87405457	97.244
manual_advapi32_10_85	0.999	3.047	0.81	99.85872836	96.953
manual_advapi32_10_90	0.995	3.68	0.856	99.8236588	96.32
manual_advapi32_10_95	0.993	4.564	0.923	99.78234111	95.436
manual_advapi32_10_100	0.993	51.452	0.998	97.57522374	48.548

“msvcrt.dll” Import Functions

Five Functions – msvcrt.dll

‘strcmp’, ‘_P__environ’, ‘_read’, ‘isctype’, and ‘memchr’

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	5,231	159,937	24,350	15.22%
train_features_1.JSONI	4,202	130,416	18,684	14.326%
train_features_2.JSONI	5,778	183,830	25,890	14.083%
train_features_3.JSONI	5,777	192,425	25,667	13.338%
train_features_4.JSONI	5,230	176,556	23,237	13.161%
train_features_5.JSONI	7,138	225,072	31,917	14.18%
Total	33,356	1,068,236	149,745	14.017%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_msvcrt_5_5	1	1.812	0.533	99.90669428	98.188
manual_msvcrt_5_10	1	1.8	0.544	99.90637742	98.2
manual_msvcrt_5_15	0.999	1.736	0.528	99.90444305	98.264
manual_msvcrt_5_20	0.998	1.784	0.53	99.90882077	98.216
manual_msvcrt_5_25	0.998	1.836	0.541	99.90876486	98.164
manual_msvcrt_5_30	0.993	1.793	0.547	99.90714237	98.207
manual_msvcrt_5_35	0.998	1.841	0.549	99.90493044	98.159
manual_msvcrt_5_40	0.997	1.831	0.546	99.9060563	98.169
manual_msvcrt_5_45	0.996	1.858	0.541	99.90589833	98.142
manual_msvcrt_5_50	0.994	1.818	0.546	99.90607513	98.182
manual_msvcrt_5_55	0.994	1.82	0.552	99.89533784	98.18
manual_msvcrt_5_60	1	1.813	0.564	99.90433013	98.187
manual_msvcrt_5_65	0.998	1.888	0.561	99.89730125	98.112
manual_msvcrt_5_70	0.998	1.948	0.574	99.89955739	98.052
manual_msvcrt_5_75	0.997	1.842	0.571	99.89618596	98.158
manual_msvcrt_5_80	0.997	2.044	0.591	99.89331908	97.956
manual_msvcrt_5_85	0.997	1.927	0.602	99.88853725	98.073
manual_msvcrt_5_90	0.998	2.262	0.638	99.87940372	97.738
manual_msvcrt_5_95	1	2.8	0.691	99.85315427	97.2
manual_msvcrt_5_100	0.997	4.114	0.802	99.75700525	95.886

Ten Functions – msvcrt.dll

'strcmp', '__p__environ', '_read', '_isctype', 'memchr', '__p__fmode', '__getmainargs', '_cexit', 'atexit', and '_assert'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	5,231	159,937	46,412	29.018%
train_features_1.JSONI	4,202	130,416	36,373	27.889%
train_features_2.JSONI	5,778	183,830	50,169	27.29%
train_features_3.JSONI	5,777	192,425	48,871	25.397%
train_features_4.JSONI	5,230	176,556	45,193	25.596%
train_features_5.JSONI	7,138	225,072	64,476	28.646%
Total	33,356	1,068,236	291,494	27.287%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
manual_msvcrt_10_5	0.998	1.864	0.542	99.90674445	98.136
manual_msvcrt_10_10	0.996	1.832	0.538	99.90611006	98.168
manual_msvcrt_10_15	0.996	1.814	0.528	99.90828948	98.186
manual_msvcrt_10_20	0.999	1.784	0.54	99.9072734	98.216
manual_msvcrt_10_25	0.997	1.804	0.535	99.90921441	98.196
manual_msvcrt_10_30	0.997	1.798	0.538	99.90191497	98.202
manual_msvcrt_10_35	0.998	1.847	0.545	99.89805365	98.153
manual_msvcrt_10_40	0.998	1.888	0.543	99.90330687	98.112
manual_msvcrt_10_45	1	1.782	0.534	99.90429628	98.218
manual_msvcrt_10_50	0.996	1.795	0.547	99.90214127	98.205
manual_msvcrt_10_55	1	1.867	0.551	99.9026623	98.133
manual_msvcrt_10_60	0.998	1.787	0.55	99.90680335	98.213
manual_msvcrt_10_65	0.999	1.886	0.558	99.90406463	98.114
manual_msvcrt_10_70	0.993	1.972	0.573	99.90437755	98.028
manual_msvcrt_10_75	0.997	1.86	0.568	99.89257589	98.14
manual_msvcrt_10_80	0.999	2.044	0.587	99.89521826	97.956
manual_msvcrt_10_85	0.998	2.123	0.611	99.88620192	97.877
manual_msvcrt_10_90	0.998	2.291	0.638	99.87449242	97.709
manual_msvcrt_10_95	0.997	2.649	0.68	99.84857023	97.351
manual_msvcrt_10_100	0.996	4.623	0.806	99.74560355	95.377

Randomly Selected Functions

“user32.dll” Import Functions

Five Functions – user32.dll

'SetCursor', 'GetSystemMetrics', 'GetDesktopWindow', 'TrackPopupMenu', 'ClientToScreen'

Training file	Unique Library Count	Library Count	Function Count	Injection Count	Injection Percentage
train_features_0.JSONI	12,824	654,156	42,939		6.564%
train_features_1.JSONI	11,664	590,103	39,515		6.696%
train_features_2.JSONI	13,655	694,225	46,239		6.660%
train_features_3.JSONI	16,678	835,414	56,176		6.724%
train_features_4.JSONI	17,940	1,313,214	48,533		3.695%
train_features_5.JSONI	11,772	511,952	42,643		8.329%
Total	84,533	4,599,064	276,045		6.002%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_user32_5_5	0.993	1.927	0.549	99.90532528	98.073
random_user32_5_10	0.999	1.784	0.537	99.904929	98.216
random_user32_5_15	0.998	1.805	0.542	99.90532947	98.195
random_user32_5_20	1	1.915	0.561	99.89477319	98.085
random_user32_5_25	0.998	1.867	0.563	99.8972514	98.133
random_user32_5_30	0.999	1.95	0.569	99.8949143	98.05
random_user32_5_35	1	1.895	0.574	99.89260108	98.105
random_user32_5_40	0.997	2.065	0.601	99.88584482	97.935
random_user32_5_45	0.999	2.025	0.603	99.88949437	97.975
random_user32_5_50	0.997	2.052	0.621	99.87827024	97.948
random_user32_5_55	1	2.211	0.639	99.87199413	97.789
random_user32_5_60	0.997	2.308	0.659	99.87576517	97.692
random_user32_5_65	1	2.331	0.68	99.8701682	97.669
random_user32_5_70	0.999	2.413	0.691	99.8624431	97.587
random_user32_5_75	0.994	2.491	0.715	99.85279526	97.509
random_user32_5_80	1	2.747	0.764	99.83588854	97.253
random_user32_5_85	0.998	3.227	0.811	99.82681944	96.773
random_user32_5_90	0.998	3.631	0.857	99.79327915	96.369
random_user32_5_95	0.99	5.934	0.931	99.71186741	94.066
random_user32_5_100	0.569	48.914	0.998	98.41913879	51.086

Ten Functions – user32.dll

'SetCursor', 'GetSystemMetrics', 'GetDesktopWindow', 'TrackPopupMenu', 'ClientToScreen',
'CloseClipboard', 'PeekMessageA', 'GetDlgItem', 'BeginPaint', 'EmptyClipboard'

Training file	Unique Library Count	Library Count	Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	12,824	654,156		87,294	13.344%
train_features_1.JSONI	11,664	590,103		81,266	13.771%
train_features_2.JSONI	13,655	694,225		95,805	13.800%
train_features_3.JSONI	16,678	835,414		115,694	13.848%
train_features_4.JSONI	17,940	1,313,214		110,488	8.413%
train_features_5.JSONI	11,772	511,952		87,960	17.181%
Total	84,533	4,599,064		578,507	12.578%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_user32_10_5	1	1.735	0.53	99.91302554	98.265
random_user32_10_10	0.996	1.785	0.54	99.90602313	98.215
random_user32_10_15	1	1.96	0.549	99.89885751	98.04
random_user32_10_20	1	1.891	0.546	99.89833258	98.109
random_user32_10_25	0.999	1.854	0.557	99.89558585	98.146
random_user32_10_30	0.996	2.028	0.573	99.88481099	97.972
random_user32_10_35	1	1.973	0.59	99.88033077	98.027
random_user32_10_40	0.998	2.001	0.602	99.88344186	97.999
random_user32_10_45	0.997	2.013	0.614	99.87800445	97.987
random_user32_10_50	0.995	2.131	0.627	99.8757583	97.869
random_user32_10_55	0.996	2.152	0.645	99.86890763	97.848
random_user32_10_60	0.997	2.221	0.664	99.87108499	97.779
random_user32_10_65	1	2.268	0.68	99.86566299	97.732
random_user32_10_70	0.995	2.396	0.71	99.85938025	97.604
random_user32_10_75	1	2.593	0.748	99.84678703	97.407
random_user32_10_80	0.994	2.946	0.788	99.83933104	97.054
random_user32_10_85	0.998	3.018	0.815	99.8203463	96.982
random_user32_10_90	1	3.905	0.878	99.78895287	96.095
random_user32_10_95	0.983	5.9	0.936	99.71879549	94.1
random_user32_10_100	0.264	60.938	0.999	98.03389002	39.062

“shell32.dll” Import Functions

Five Functions – shell32.dll

'ExtractIconW', 'SHGetDesktopFolder', 'FreeIconList', 'ExtractAssociatedIconExW',
'DllRegisterServer'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	7,187	33,299	35,044	105.240%
train_features_1.JSONI	6,304	28,970	30,545	105.436%
train_features_2.JSONI	7,447	34,450	36,046	104.632%
train_features_3.JSONI	9,537	40,372	46,617	115.468%
train_features_4.JSONI	11,584	78,278	50,252	64.196%
train_features_5.JSONI	5,691	26,453	27,576	104.245%
Total	47,750	241,822	226,080	93.490%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_shell32_5_5	0.999	1.782	0.531	99.91016627	98.218
random_shell32_5_10	0.995	1.875	0.54	99.90711866	98.125
random_shell32_5_15	0.998	1.805	0.543	99.90723896	98.195
random_shell32_5_20	1	1.86	0.543	99.91042568	98.14
random_shell32_5_25	0.996	1.797	0.551	99.90494492	98.203
random_shell32_5_30	1	1.87	0.557	99.90508319	98.13
random_shell32_5_35	0.998	1.907	0.575	99.90237536	98.093
random_shell32_5_40	0.997	1.99	0.579	99.90400275	98.01
random_shell32_5_45	0.999	1.977	0.592	99.90038427	98.023
random_shell32_5_50	0.997	1.999	0.603	99.89790463	98.001
random_shell32_5_55	0.998	2.055	0.614	99.89354255	97.945
random_shell32_5_60	0.997	2.049	0.628	99.89605343	97.951
random_shell32_5_65	0.996	2.283	0.646	99.88918586	97.717
random_shell32_5_70	0.997	2.455	0.674	99.88255586	97.545
random_shell32_5_75	0.998	2.668	0.698	99.87591172	97.332
random_shell32_5_80	1	2.641	0.72	99.87056884	97.359
random_shell32_5_85	1	3.12	0.761	99.8526714	96.88
random_shell32_5_90	0.999	3.308	0.803	99.83860904	96.692
random_shell32_5_95	0.999	5.068	0.896	99.77321202	94.932
random_shell32_5_100	0.944	21.954	0.991	99.15268717	78.046

Ten Functions – shell32.dll

'ExtractIconW', 'SHGetDesktopFolder', 'FreeIconList', 'ExtractAssociatedIconExW',
 'DllRegisterServer', 'ShellMessageBoxA', 'ShellAboutW', 'ExtractIconExW',
 'SHBrowseForFolderA', 'SHCreateProcessAsUserW'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	7,187	33,299	68,775	206.537%
train_features_1.JSONI	6,304	28,970	60,872	210.120%
train_features_2.JSONI	7,447	34,450	71,714	208.168%
train_features_3.JSONI	9,537	40,372	92,215	228.413%
train_features_4.JSONI	11,584	78,278	106,198	135.667%
train_features_5.JSONI	5,691	26,453	54,884	207.477%
Total	47,750	241,822	454,658	188.013%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_shell32_10_5	1	1.734	0.53	99.90545204	98.266
random_shell32_10_10	1	1.849	0.545	99.90021741	98.151
random_shell32_10_15	0.998	1.795	0.549	99.9145051	98.205
random_shell32_10_20	0.997	1.867	0.54	99.90825568	98.133
random_shell32_10_25	1	1.832	0.554	99.90514235	98.168
random_shell32_10_30	0.999	1.862	0.559	99.89999392	98.138
random_shell32_10_35	1	1.924	0.57	99.90513604	98.076
random_shell32_10_40	0.999	1.871	0.578	99.90597338	98.129
random_shell32_10_45	1	2.026	0.593	99.89424616	97.974
random_shell32_10_50	0.995	2.08	0.614	99.89824944	97.92
random_shell32_10_55	0.997	2.122	0.609	99.89443564	97.878
random_shell32_10_60	0.996	2.232	0.635	99.88967818	97.768
random_shell32_10_65	1	2.327	0.644	99.88521883	97.673
random_shell32_10_70	0.996	2.388	0.671	99.87998321	97.612
random_shell32_10_75	0.997	2.547	0.702	99.87900054	97.453
random_shell32_10_80	0.998	2.596	0.722	99.87299072	97.404
random_shell32_10_85	0.998	2.968	0.76	99.85894896	97.032
random_shell32_10_90	0.991	3.576	0.817	99.83345983	96.424
random_shell32_10_95	0.991	4.8	0.89	99.77162415	95.2
random_shell32_10_100	0.944	21.954	0.991	99.15268717	78.046

“advapi32.dll” Import Functions

Five Functions – advapi32.dll

'CryptAcquireContextA', 'AdjustTokenPrivileges', 'SetSecurityDescriptorDacl',
'GetSecurityDescriptorDacl', 'UnlockServiceDatabase'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	12,576	159,639	56,433	35.350%
train_features_1.JSONI	15,956	237,016	68,782	29.019%
train_features_2.JSONI	16,272	227,814	70,998	31.164%
train_features_3.JSONI	14,039	159,156	64,071	40.256%
train_features_4.JSONI	15,801	217,846	73,433	33.708%
train_features_5.JSONI	9,910	116,957	44,959	38.440%
Total	84,554	1,118,428	378,676	33.857%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_advapi32_5_5	0.998	1.747	0.533	99.90913011	98.253
random_advapi32_5_10	0.998	1.781	0.544	99.90669201	98.219
random_advapi32_5_15	1	1.894	0.563	99.90528096	98.106
random_advapi32_5_20	1	1.917	0.555	99.90621429	98.083
random_advapi32_5_25	1	1.941	0.56	99.90894943	98.059
random_advapi32_5_30	0.995	1.879	0.581	99.90207243	98.121
random_advapi32_5_35	0.998	1.872	0.588	99.90633421	98.128
random_advapi32_5_40	0.999	1.994	0.598	99.90085319	98.006
random_advapi32_5_45	0.999	1.777	0.605	99.90421925	98.223
random_advapi32_5_50	1	1.971	0.626	99.90039098	98.029
random_advapi32_5_55	0.998	2.038	0.645	99.90303338	97.962
random_advapi32_5_60	0.999	1.977	0.644	99.89555245	98.023
random_advapi32_5_65	1	2.078	0.678	99.8968961	97.922
random_advapi32_5_70	0.993	2.202	0.704	99.89492523	97.798
random_advapi32_5_75	0.997	2.373	0.726	99.89142925	97.627
random_advapi32_5_80	0.997	2.757	0.769	99.87881958	97.243
random_advapi32_5_85	1	2.883	0.794	99.8619755	97.117
random_advapi32_5_90	0.99	3.542	0.866	99.83851064	96.458
random_advapi32_5_95	0.994	5.588	0.93	99.77036701	94.412
random_advapi32_5_100	0.539	63.209	0.999	96.91026982	36.791

Ten Functions – advapi32.dll

'CryptAcquireContextA', 'AdjustTokenPrivileges', 'SetSecurityDescriptorDacl',
 'GetSecurityDescriptorDacl', 'UnlockServiceDatabase', 'GetAce', 'EqualSid', 'InitializeAcl',
 'CloseServiceHandle', 'CryptAcquireContextW'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	12,576	159,639	114,085	71.464%
train_features_1.JSONI	15,956	237,016	139,051	58.667%
train_features_2.JSONI	16,272	227,814	143,771	63.108%
train_features_3.JSONI	14,039	159,156	129,613	81.437%
train_features_4.JSONI	15,801	217,846	145,436	66.760%
train_features_5.JSONI	9,910	116,957	91,261	78.029%
Total	84,554	1,118,428	763,217	68.240%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_advapi32_10_5	0.999	1.809	0.538	99.90965663	98.191
random_advapi32_10_10	0.998	1.807	0.544	99.90645946	98.193
random_advapi32_10_15	0.997	1.837	0.552	99.90745487	98.163
random_advapi32_10_20	0.997	1.805	0.562	99.91021818	98.195
random_advapi32_10_25	0.997	1.925	0.576	99.90211576	98.075
random_advapi32_10_30	0.995	1.834	0.565	99.90755966	98.166
random_advapi32_10_35	0.999	1.988	0.586	99.89921748	98.012
random_advapi32_10_40	0.998	1.863	0.595	99.908428	98.137
random_advapi32_10_45	1	1.887	0.607	99.90483515	98.113
random_advapi32_10_50	0.998	1.987	0.62	99.90047496	98.013
random_advapi32_10_55	0.995	1.833	0.632	99.89978298	98.167
random_advapi32_10_60	0.998	1.84	0.645	99.90276438	98.16
random_advapi32_10_65	0.997	2.127	0.671	99.89205745	97.873
random_advapi32_10_70	1	2.411	0.69	99.89050188	97.589
random_advapi32_10_75	0.992	2.286	0.726	99.88938385	97.714
random_advapi32_10_80	0.994	2.485	0.765	99.88257789	97.515
random_advapi32_10_85	0.993	3.015	0.804	99.8610278	96.985
random_advapi32_10_90	0.994	3.284	0.851	99.84825922	96.716
random_advapi32_10_95	0.986	5.378	0.927	99.77633088	94.622
random_advapi32_10_100	0.353	65.769	0.999	97.22543733	34.231

“msvcrt.dll” Import Functions

Five Functions – msvcrt.dll

'strftime', '_iob', '_assert', '___dllonexit', 'memcmp'

Training file	Unique Count	Library	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	5,231		159,937	20,883	13.057%
train_features_1.JSONI	4,202		130,416	15,103	11.580%
train_features_2.JSONI	5,778		183,830	21,254	11.561%
train_features_3.JSONI	5,777		192,425	21,331	11.085%
train_features_4.JSONI	5,230		176,556	18,981	10.750%
train_features_5.JSONI	7,138		225,072	24,429	10.853%
Total	33,356		1,068,236	121,981	11.418%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_msvcrt_5_5	0.998	1.763	0.525	99.90527376	98.237
random_msvcrt_5_10	0.998	1.763	0.525	99.90527342	98.237
random_msvcrt_5_15	0.998	1.755	0.528	99.9073185	98.245
random_msvcrt_5_20	1	1.811	0.529	99.90563236	98.189
random_msvcrt_5_25	0.997	1.854	0.534	99.9070214	98.146
random_msvcrt_5_30	1	1.807	0.529	99.90367091	98.193
random_msvcrt_5_35	0.997	1.695	0.533	99.91055077	98.305
random_msvcrt_5_40	0.994	1.844	0.538	99.90713388	98.156
random_msvcrt_5_45	1	1.774	0.536	99.90566008	98.226
random_msvcrt_5_50	0.999	1.782	0.54	99.90566768	98.218
random_msvcrt_5_55	0.998	1.838	0.543	99.90597869	98.162
random_msvcrt_5_60	0.995	1.888	0.545	99.90630827	98.112
random_msvcrt_5_65	0.996	1.8	0.55	99.90256226	98.2
random_msvcrt_5_70	1	1.961	0.553	99.8986999	98.039
random_msvcrt_5_75	0.998	1.885	0.548	99.903245	98.115
random_msvcrt_5_80	0.998	1.896	0.565	99.90174819	98.104
random_msvcrt_5_85	1	1.988	0.578	99.89862215	98.012
random_msvcrt_5_90	1	2.069	0.594	99.88850946	97.931
random_msvcrt_5_95	1	2.426	0.625	99.87477996	97.574
random_msvcrt_5_100	0.998	2.876	0.689	99.84286116	97.124

Ten Functions – msvcrt.dll

'strftime', '_iob', '_assert', '__dllonexit', 'memcmp', '_controlfp', 'localeconv', 'memchr',
'malloc', '??1type_info@@@UAE@XZ'

Training file	Unique Library Count	Library Function Count	Function Injection Count	Injection Percentage
train_features_0.JSONI	5,231	159,937	40,027	25.026%
train_features_1.JSONI	4,202	130,416	30,581	23.448%
train_features_2.JSONI	5,778	183,830	42,601	23.174%
train_features_3.JSONI	5,777	192,425	42,096	21.876%
train_features_4.JSONI	5,230	176,556	37,674	21.338%
train_features_5.JSONI	7,138	225,072	50,678	22.516%
Total	33,356	1,068,236	243,657	22.809%

Attack Name	FPR 1%	FNR 1%	Threshold 1%	ROC	Detection Rate 1%
random_msvcrt_10_5	0.998	1.854	0.53	99.90285945	98.146
random_msvcrt_10_10	0.998	1.772	0.532	99.90829745	98.228
random_msvcrt_10_15	1	1.73	0.518	99.90968703	98.27
random_msvcrt_10_20	0.996	1.684	0.523	99.90835256	98.316
random_msvcrt_10_25	0.993	1.737	0.524	99.90470328	98.263
random_msvcrt_10_30	1	1.773	0.528	99.90857437	98.227
random_msvcrt_10_35	0.999	1.764	0.532	99.90814562	98.236
random_msvcrt_10_40	0.998	1.798	0.551	99.89949646	98.202
random_msvcrt_10_45	0.996	1.888	0.543	99.90375495	98.112
random_msvcrt_10_50	0.999	1.816	0.552	99.90200585	98.184
random_msvcrt_10_55	0.995	1.897	0.553	99.8973074	98.103
random_msvcrt_10_60	0.999	1.877	0.555	99.90479969	98.123
random_msvcrt_10_65	0.999	1.935	0.558	99.90049668	98.065
random_msvcrt_10_70	0.997	1.839	0.56	99.9013925	98.161
random_msvcrt_10_75	0.995	1.947	0.585	99.89818232	98.053
random_msvcrt_10_80	0.998	1.844	0.578	99.89975858	98.156
random_msvcrt_10_85	0.999	2.242	0.615	99.88516363	97.758
random_msvcrt_10_90	1	2.34	0.63	99.87929476	97.66
random_msvcrt_10_95	0.999	2.766	0.677	99.85757581	97.234
random_msvcrt_10_100	0.998	4.346	0.786	99.76336991	95.654