Graduate Theses, Dissertations, and Problem Reports

2019

Intelligent Malware Detection Using File-to-file Relations and Enhancing its Security against Adversarial Attacks

Lingwei Chen
lgchen@mix.wvu.edu

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Part of the Information Security Commons

Recommended Citation

Chen, Lingwei, "Intelligent Malware Detection Using File-to-file Relations and Enhancing its Security against Adversarial Attacks" (2019). *Graduate Theses, Dissertations, and Problem Reports*. 3844.
https://researchrepository.wvu.edu/etd/3844

This Dissertation is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Dissertation in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Dissertation has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Intelligent Malware Detection Using File-to-file Relations and Enhancing its Security against Adversarial Attacks

Lingwei Chen

Dissertation submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Yanfang Ye, Ph.D., Committee Chairperson
Donald Adjeroh, Ph.D.
Elaine M. Eschen, Ph.D.
Zachariah B. Etienne, Ph.D.
Katerina Goseva-Popstojanova, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2019

Keywords: Malware Detection, Machine Learning, Data Mining, File-to-file Relations, Adversarial Attacks and Defenses

Copyright 2019 Lingwei Chen

Abstract

Intelligent Malware Detection Using File-to-file Relations and Enhancing its Security against Adversarial Attacks

Lingwei Chen

With computing devices and the Internet being indispensable in people's everyday life, malware has posed serious threats to their security, making its detection of utmost concern. To protect legitimate users from the evolving malware attacks, machine learning-based systems have been successfully deployed and offer unparalleled flexibility in automatic malware detection. In most of these systems, resting on the analysis of different content-based features either statically or dynamically extracted from the file samples, various kinds of classifiers are constructed to detect malware. However, besides content-based features, file-to-file relations, such as file co-existence, can provide valuable information in malware detection and make evasion harder. To better understand the properties of file-to-file relations, we construct the file co-existence graph. Resting on the constructed graph, we investigate the semantic relatedness among files, and leverage graph inference, active learning and graph representation learning for malware detection. Comprehensive experimental results on the real sample collections from Comodo Cloud Security Center demonstrate the effectiveness of our proposed learning paradigms.

As machine learning-based detection systems become more widely deployed, the incentive for defeating them increases. Therefore, we go further insight into the arms race between adversarial malware attack and defense, and aim to enhance the security of machine learning-based malware detection systems. In particular, we first explore the adversarial attacks under different scenarios (i.e., different levels of knowledge the attackers might have about the targeted learning system), and define a general attack strategy to thoroughly assess the adversarial behaviors. Then, considering different skills and capabilities of the attackers, we propose the corresponding secure-learning paradigms to counter the adversarial attacks and enhance the security of the learning systems while not compromising the detection accuracy. We conduct a series of comprehensive experimental studies based on the real sample collections from Comodo Cloud Security Center and the promising results demonstrate the effectiveness of our proposed secure-learning models, which can be readily applied to other detection tasks.

Acknowledgments

It is never easy to finish the Ph.D. study and write this dissertation. It would not have been possible to finish without the help of so many people in many ways.

I would first like to express my greatest gratitude to my committee chair and advisor, Dr. Yanfang Ye, for her guidance and support not only for this dissertation but throughout the time of my entire Ph.D. study. Her passion, vision, attitude, and love for research is always an inspiration source and influence to me; her expertise, understanding, generous guidance, suggestions, valuable comments and revisions make it possible for me to work on such an exciting topic; her devotion of significant time and efforts on mentoring my research has resulted in fourteen publications by the date of this dissertation. In particular, our work on adversarial machine learning in malware detection has resulted in several high quality publications, including the prestigious *IEEE EISIC'2017 Best Paper Award* and recent publicaitons in top-tier conferences (e.g., *ACSAC'2017 with 19.7% acceptance rate*, and *ACSAC'2018 with 20.1% acceptance rate*).

I would also like to thank my committee members, Dr. Donald Adjeroh, Dr. Elaine Eschen, Dr. Katerina Goseva-Popstojanova, and Dr. Zachariah Etienne, for their time and help for my research work; I am very fortunate to work with a cheerful group members, including Shifu Hou, Yujie Fan, Yiming Zhang, Jian Liu, William Harday and Aaron Saas, who exchanged ideas about cybersecurity and machine learning related research work and provided useful suggestions on my dissertation.

I am highly thankful to be blessed by amazing and talented family members and friends, who have made such a positive impact on my daily life, study, and research; last but not the least, I would also like to thank the anti-malware experts of Comodo Security Lab for data collection and helpful discussion. This work is partially supported by the U.S. National Science Foundation under grants CNS-1618629, CNS-1814825 and OAC-1839909, WV Higher Education Policy Commission Grant (HEPC.dsr.18.5), and WVU Research and Scholarship Advancement Grant (R-844).

Contents

Acknowledgments **iii**

List of Figures **vi**

List of Tables **ix**

List of Notations **x**

List of Acronyms **xii**

1 Introduction **1**
- 1.1 Background and Motivation . . . 1
- 1.2 Research Goals . . . 3
- 1.3 Contributions of This Dissertation . . . 4
- 1.4 Organization of This Dissertation . . . 7

2 Development of Malware and Malware Detection Techniques **8**
- 2.1 Development of Malware . . . 8
  - 2.1.1 Definition of Malware . . . 8
  - 2.1.2 Taxonomy of Malware . . . 9
  - 2.1.3 History of Malware . . . 11
- 2.2 Development of Malware Detection Techniques . . . 13
  - 2.2.1 Signature-based Malware Detection . . . 13
  - 2.2.2 Heuristic-based Malware Detection . . . 14
  - 2.2.3 Machine Learning-based Malware Detection . . . 15
  - 2.2.4 Adversarial Machine Learning in Malware Detection . . . 17

3 Intelligent Malware Detection Utilizing File-to-file Relations **20**
- 3.1 File-to-file Relation Graph Construction . . . 21
- 3.2 An Enhanced Belief Propagation Algorithm for Malware Detection . . . 24
  - 3.2.1 Standard Belief Propagation . . . 25
  - 3.2.2 Enhanced Belief Propagation . . . 26
  - 3.2.3 Experimental Results and Analysis . . . 28
- 3.3 Active Learning in Malware Detection . . . 32
  - 3.3.1 Gaining Insight into the Semantic Relatedness . . . 32
  - 3.3.2 Active Learning Framework . . . 38
  - 3.3.3 Experimental Results and Analysis . . . 40
- 3.4 Graph Representation Learning for Malware Detection . . . 44

3.4.1 Representation Learning using Long Short-term Memory 44
3.4.2 Experimental Results and Analysis 48
3.5 Summary 50

4 Enhancing Security of Learning-based Systems in Malware Detection 52
4.1 Problem Definition 53
4.2 Adversarial Attack 54
4.2.1 Feature Manipulation 56
4.2.2 Adversarial Cost 57
4.2.3 Attack Strategy 58
4.3 *SecDefender*: A Secure-learning Model against Well-crafted Attack 59
4.3.1 Feature Representation 59
4.3.2 Well-crafted Attack Model *AdvAttack* 61
4.3.3 Secure-learning Model based on *AdvAttack* 65
4.3.4 Experimental Results and Analysis 68
4.4 *SecureDroid*: A Secure-learning Paradigm against Various Kinds of Attacks 73
4.4.1 Feature Representation 74
4.4.2 Secure Classifier Construction using Novel feature Selection 76
4.4.3 Ensemble Learning to Improve Detection Accuracy 80
4.4.4 Experimental Results and Analysis 82
4.5 *DroidEye*: Fortifying Learning Security over Feature Space 90
4.5.1 Feature Representation 91
4.5.2 Count Featurization 92
4.5.3 Experimental Results and Analysis 96
4.6 Summary 101

5 Conclusion and Future Work 104
5.1 Conclusion 104
5.2 Future Work 106

List of Publications 109

Bibliography 111

List of Figures

1.1 File relations between a Downloader-Trojans and its related Trojans [149] 2
1.2 File relations between a benign application and its related dynamic link files [149] 2

2.1 (a) New malware created in the last two years, and (b) total malware created in the last ten years [4]. 9

3.1 An example of file-to-file relation graph. 21
3.2 Visualization of file-to-file relation graphs: (a) a part of the constructed graph; (b) an example of a malware relation graph with one-hop information; (c) an example of a benign file relation graph with one-hop information (Red nodes denote malware, green nodes represent benign file, and yellow nodes are unknown file) [27]. 22
3.3 A zoom-in view of a part of the constructed file relation graph [66]. 23
3.4 Message update from node i to node j 25
3.5 A sample dataset and its file relation graph constructed 26
3.6 Comparisons of different belief propagation algorithms 30
3.7 Comparisons of malware detection effectiveness and efficiency between EBP Algorithm and other classification approaches. 31
3.8 Malare detection comparisons using large and real data collection. 32
3.9 Indirect influences superior than direct influences for file 1880 (yellow node) 35
3.10 The comparison of benign files and malware in IoB and IoM measures . . 35
3.11 The comparisons of “important” malware and “non-important” ones. . . . 36
3.12 Graph structure comparisons of “important” and “non-important” malware. (a) Important malware A and its neighbors; (b) Relations between A’s neighbors; (c) Non-important malware B and its neighbors; (d) Relations between B’s neighbors [27]. 37
3.13 An example of malware detection using active learning framework. 42
3.14 Comparisons of ROC curves and detection efficiency of different methods 43
3.15 Neighborhood relationships among files. 45
3.16 Illustration of encoder-decoder LSTM architecture. 46
3.17 Parameter sensitivity evaluation. 50

4.1 Intelligent malware detection system using machine learning techniques. . 53

4.2 Different scenarios of the adversarial attacks. With the direction of the inward arrow, the adversarial attacks are depicted with the knowledge of $(X, \hat{D})$, (X, D), and (X, D, f). 56
4.3 An overview of system architecture of *SecDefender*. In this system, the collected PE files are first represented as d-dimensional binary feature vectors. Then a well-crafted adversarial attack model *AdvAttack* is formulated to generate the adversarial examples, which will be further used for classifier retraining and security regularization. For a new file, based on the extracted features, it will be predicted as either malicious or benign based on the trained classification model. 59
4.4 Relevance score distribution of the extracted API calls for the classification of malware and benign files 61
4.5 The feature distribution of file samples. 68
4.6 Comparisons of *AdvAttack* and other adversarial attacks: *Original-Classifier* (0), different adversarial attacks (Method 1 - 4) and *AdvAttack* (5) 69
4.7 *FNR* before and after each attack under different scenarios for all 1,000 testing file samples 70
4.8 Comparisons of *SecDefender* and other classification models on *ACC*, *F1*, *FNR*, and ROC curves: *Original-Classifier* (1), *Original-Classifier* under attack (2), retrained *Original-Classifier* (3), and *SecDefender* (4) 71
4.9 An overview of system architecture of *SecureDroid*. In the system, the collected app files are first represented as d-dimensional binary feature vectors. Then *SecCLS* is applied to select a set of features (each feature i is selected with probability $\mathcal{P}(i)$) to construct a more secure classifier. *SecENS* is later exploited to aggregate different individual classifiers built using *SecCLS* to classify malicious and benign files. For a new file, based on the extracted features, it will be predicted as either malicious or benign based on the trained classification model. 74
4.10 The manipulation costs determined by different feature types and manipulation methods. 78
4.11 Effectiveness evaluation of different attacks. 84
4.12 Security evaluations under brute-force (BF) attacks, anonymous (AN) attacks, well-crafted (WC) attacks, and without attacks. 86
4.13 Comparisons of different defense methods. 88
4.14 Scalability and stability evaluation of *SecureDroid*. 89
4.15 An overview of system architecture of *DroidEye*. In the system, the collected apps are first represented as d-dimensional binary feature vectors. To harden the evasion, *count featurization* is used to transform each binary feature vector $\mathbf{x}_i$ to a continuous feature vector $\mathbf{x}'_i$; then *softmax function with adversarial parameter* is introduced to find the best trade-off between security and accuracy for the classifier. For a new app, after feature representation, it will be predicted as either benign or malicious using the classifier. 90
4.16 Defenses in different feature spaces. 92
4.17 An example of count featurization. 94

4.18 Evaluation of *DroidEye* with different τ under AdvAttack with number of manipulated features varying from 10 to 50. 98
4.19 Security evaluations of *DroidEye* and *Original-Classifier* under AdvAattack, FGSM attack, ANAattack, and without attacks. 99
4.20 Comparisons of different defense methods. 101

List of Tables

3.1 Graph property comparisons 24
3.2 The edge potential design in AESOP[121] 26
3.3 The results of standard BP and EBP based on Figure 3.5 27
3.4 The edge potential design in enhanced BP 27
3.5 The evaluation measures of malware detection performance 30
3.6 Malare detection comparisons using large and real data collection 31
3.7 Evaluation of the designed graph-based features 41
3.8 Evaluation of the proposed learning framework in malware detection . . . 42
3.9 Comparisons of different detection methods 43
3.10 Comparisons of *file2vec* with DeepWalk in malware detection 48
3.11 Comparisons with other machine learning methods 49

4.1 List of the top ranked API calls 62
4.2 Comparisons of different anti-malware scanners 72
4.3 Illustration of extracted features for Android apps in *SecureDroid* 76
4.4 Performance indices of Android malware detection 83
4.5 Comparison of *SecureDroid* with *SecCLS* and *ERFS* with random feature selection against well-crafted attacks (UnderAtt) and without attacks (NonAtt). 87
4.6 Illustration of extracted features for Android apps in *DroidEye* 91

List of Notations

$\|.\|$	The size of a set
$\boldsymbol{\xi}$	Lagrange multiplier
Ψ	Knowledge of the learning system
τ	Adversarial parameter
$\boldsymbol{\theta}$	Adversarial attack
δ_b^v	Vertex v's benign neighbors
δ_m^v	Vertex v's malicious neighbors
$\mathcal{A}(\mathbf{x})$	Manipulation function
$\mathcal{B}$	API calls highly relevant to benign files
$b_i(v_i)$	Belief value of node i
C_i	The set of clients containing v_i
$\mathcal{C}(\mathbf{x}', \mathbf{x})$	The adversarial cost function
$con(v_i, v_i)$	The connectivity between file v_i and v_j
c_t	Cell activation vectors
D	Training sample set
d	Vector dimension
E	The set of relations between file samples
e^v	Edges built by all v's neighbors
F_{v_i}	Concatenation of Relation- and Graph-based feature of v_i
f	Classification function
$f_{i \to j}(v_i, v_j)$	The edge potential from node i to node j
f_t	Forget gate at timestep t
G	The file relation graph
G_{v_i}	Graph-based feature of v_i
$g_i(v_i)$	The node potential of node i
$g(u, v)$	The distance between vertex u and v

$\mathcal{H}$	Hidden layer function
$\mathbf{h}_t$	The hidden layer vector at timestep t
$\mathcal{I}(i)$	The importance of ith-feature
$I(v_i, v_j)$	Indicator to denote if $(v_i, v_j) \in E$
$I(x, +1)$	Feature x's relevance score to malware
$I(x, -1)$	Feature x's relevance score to benign file
i_t	Input gate at timestep t
k_v	Degree of the vertex v
l	Walk length
$\overrightarrow{\vec{M}^t}$	The malicious score vector at timestep t
$\mathcal{M}$	API calls highly relevant to malware
$M(v_i)$	The malicious score of node v_i
$m_{i \to j}(v_j)$	The message sent from node i to node j
$N(i)$	The set of nodes neighboring node i
o_t	Output gate at timestep t
$\mathcal{P}(i)$	The probability of ith-feature being selected
R_{v_i}	Relation-based feature of v_i
r	Walks per nodes
S	The set of file states
$\mathbf{S}$	The security matrix
s_b	Benign
s_m	Malicious
s_u	Unknown
$sim(F_{v_i}, F_{v_j})$	The similarity between v_i and v_j over the feature space F
$\mathcal{T}(\mathbf{x}', \mathbf{x})$	The adversarial action function
$\mathbf{T}$	The adversarial action matrix
V	The set of file samples
v_i	File sample i
$\mathbf{v}_t$	The one-hop vector at timestep t
$w(v_i)$	The weight of the node v_i
$w(v_i, v_j)$	The weight of the edge between v_i and v_j
X	The feature space

List of Acronyms

ACC	Accuracy - (TP+TN)/(TP+TN+FP+FN)
API	Application Programming Interface
BP	Belief Propagation
CC	Closeness Centrality
CDF	Cumulative Distribution Function
DoB	Degree of Benign files
DoM	Degree of Malware
DC	Degree Centrality
DT	Decision Tree
EBP	Enhanced Belief Propagation
FGSM	Fast Gradient Sign Method
FN	False Negative - number of files (apps) mistakenly classified as benign
FNR	False Negative Rate - FN/(TP+FN)
F1	F1 Measure - 2×Precision×Recall/(Precision+Recall)
FP	False Positive - number of files (apps) mistakenly classified as malicious
FPR	False Positive Rate - FP/(FP+TN)
IDK	Ideal Knowledge Attack
IoB	Influence Coefficient of Benign files
IoM	Influence Coefficient of Malware
IPK	Imperfect Knowledge Attack
LLC	Local Clustering Coefficient
LR	Logistic Regression
LSTM	Long Short-Term Memory
MMC	Mimicry Attack

MRS	Malicious Relevance Score
Mr.SPA	Malicious Relevance Score Propagation Algorithm
MSIA	Malicious Score Inference Algorithm
NB	Naïve Bayes
NN	Neural Network
PE	Portable Executable file
Precision	Precision - TP/(TP+FP)
Recall	Recall - TP/(TP+FN)
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TN	True Negative - number of files (apps) correctly classified as benign
TNR	True Negative Rate - TN/(FP+TN)
TP	True Positive - number of files (apps) correctly classified as malicious
TPR	True Positive Rate - TP/(TP+FN)

Chapter 1

Introduction

1.1 Background and Motivation

Malware (e.g., viruses, worms, trojans, backdoors, botnets, ransomware) is ***mal***icious soft***ware*** that is disseminated by attackers as a major weapon to launch a wide range of security attacks, such as disturbing system operations, stealing personal sensitive information without user's permission, hijacking devices remotely to deliver massive spam emails, or infiltrating user's online account credentials [144]. With computing devices and the Internet being essential in everyday life, malware poses serious and evolving threats to their security, which present various damages and significant financial loss to Internet users. A study conducted by Kaspersky Lab revealed that nearly half of Internet users have encountered malicious software, in which 80% malware attacks caused problems for the users [74]. It's also reported that up to one billion dollars were stolen in roughly two years from financial institutions worldwide, due to malware attacks [73]. As a result, the detection of malware is of major concern to both the anti-malware industry and scientific research community.

In order to combat the evolving malware attacks and protect legitimate users from these threats, most malware detection systems in computing devices, especially anti-malware software products (e.g., Symantec, Kaspersky, Comodo), typically use the signature-based method [49]. A signature is a short sequence of bytes unique to each known malware, which allows newly encountered files to be correctly identified with a small error rate [75]. However, driven by economic benefits, today's malware are created at a rate of hundreds of thousands per day [149] (i.e., more than 260 million new malware samples were created last two year [4]). Meanwhile, malware attackers easily evade this method through techniques such as obfuscation, polymorphism, metamor-

phism, and encryption [119]. In order to remain effective, new and intelligent malware detection techniques need to be investigated. As a result, many research efforts have been conducted on applying machine learning techniques for intelligent malware detection [140, 7, 90, 139, 142, 95, 60, 39, 67, 68, 151, 131, 132, 137, 69]. In these systems, based on different feature representations (e.g., binary n-grams [7], system call graphs [101, 67], dynamic behaviors [49, 132], or Application Programming Interface (API) call blocks [68]), various kinds of classification approaches, such as support vector machine [153, 79, 44, 114], random forest [1] and deep neural network [68, 67, 60], are used for model construction to detect malicious files, which have offered unparalleled flexibility in intelligent malware detection.

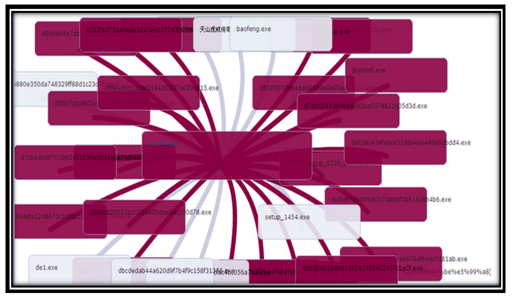


Figure 1.1: File relations between a Downloader-Trojans and its related Trojans [149]

Figure 1.2: File relations between a benign application and its related dynamic link files [149]

Most of the existing systems using machine learning techniques merely utilize local features either statically or dynamically extracted from the file samples to detect malware. As the moral says "man is known by the company he keeps", in malware detection, a file's goodness or malice may be judged by the other files that are always associated with it in the different ways. For example, Ye et al. [149] first proposed to utilize file relations for malware detection: as shown in Figure 1.1, we can observe that a Trojan-Downloader "yy(1).exe", which can download and install multiple unwanted applications (e.g., trojan, adware) from remote servers, is associated with many trojans which are marked as purple color; from Figure 1.2, we can observe that a benign system diagnostic application "everest.exe", is associated with different benign files marked in green color. Ignoring the relations among file samples is a significant limitation of current malware detection methods. Recently, features beyond file content are starting to be leveraged to curb the security threats that malware poses [149, 25, 121, 72], such as machine-to-file relations [25] and file-to-file relations (e.g., file co-existence) [149, 121], which provide

invaluable insight about the properties of file samples [149]. However, much needs to be done to take full advantage of the relationships of malware and benign files (i.e., malware-malware, malware-benign, benign-benign relations). To better understand the properties of file-to-file relations, we'd like to take a further step to delve deeper into the relationship characteristics of malware and benign files.

Meanwhile, as machine learning-based detection systems become more widely deployed, the adversary incentive for defeating them increases [144]. More specifically, machine learning itself may open the possibility for an adversary who maliciously "mis-trains" a classifier (e.g., by changing data distribution or feature importance) in a malware detection system. When the learning system is deployed in a real-world environment, it is of a great interest for malware attackers to actively manipulate the data to make the classifier produce minimum true positive (i.e., maximally misclassifying malware as benign), using some combination of prior knowledge, observation, and experimentation [38]. If we look at the evolution of malware detection techniques [49, 119, 101, 39, 25, 149, 60], malware attackers and anti-malware defenders are actually engaged in a never-ending arms race. At each round, both the malware attackers and defenders analyze the vulnerabilities of each other, and develop their own optimal strategies to overcome the opponents [14], which has led to considerable countermeasures of variability and sophistication between attackers and defenders. For example, when signature-based methods prevailed in malware detection, attackers began to use code obfuscation and encryption to bypass the detection and defeat attempts to analyze their inner mechanisms [77, 67]. Currently, the issues of understanding machine learning security in adversarial settings [152] are starting to be leveraged, from either adversarial [87, 38, 126, 14, 13] or defensive [152, 127, 40, 11, 15] perspectives. However, the application of adversarial machine learning into malware detection domain has been scarce. With machine learning techniques prevailing in malware detection, such adversaries will become even more outrageous.

1.2 Research Goals

With the above limitations and challenges addressed for intelligent malware detection, in this Ph.D. dissertation, we focus on investigating file-to-file relations to facilitate analysis of malware detection, and exploring the adversarial machine learning to enhance the security of malware detection. Specifically, we describe these research goals (*RG1-RG2*) in detail as follows:

- *RG1: Intelligent malware detection utilizing file-to-file relations:* To achieve our research goals, we construct the file co-existence graphs between malware and benign files, and analyze effective graph-based feature representations and relationship characteristics (e.g., mutual influence and difference importances among the files) for intelligent malware detection. Since feeding the graph-based features to the traditional machine learning-based classifiers is an inferior fashion for malware detection, which is incapable of depicting the file-to-file relations, a well designed graph inference framework or graph representation learning framework should be proposed to make the best use of graph structure, and manage to propagate the mutual information between malware and benign files in the constructed relationship graphs to promote the optimal solution for file labeling.

- *RG2: Enhancing security of machine learning-based malware detection:* In the arms race between malware attackers and defenders, they utilize the vulnerabilities of each other and implement their optimal strategies to overcome the opponents. To be resilient against the malware attacks, we analyze the general adversarial strategy to facilitate assessing the security of the classifier. Accordingly, we present several secure-learning paradigms to counter adversarial attacks. To be feasible in practical use for malware detection, in these paradigms, we formulate some models based on the attackers' skills and capabilities, while others are independent from the knowledge about the structure of the data (e.g., adversarial examples) or the attack model, which are adaptive to all potential attacks without exhibiting significant evidence of manipulation.

1.3 Contributions of This Dissertation

In this dissertation, to address the aforementioned research problems, we conduct feasibility studies, propose different learning methods for intelligent malware detection and adversarial machine learning, and develop the corresponding systems that integrate our proposed methods. The contributions can be summarized as follows:

- We utilize file-to-file relations for intelligent malware detection through constructing file co-existence graph construction, designing graph-based features to characterize the semantic relatedness among them, and proposing graph learning framework to detect malware, which have resulted in 4 publications [31, 27, 66, 143].

- *Deep analysis of file-to-file relation graphs:* Different from file content based detection, we analyze and utilize the relations among file samples (i.e., co-existences of the files) collected from the user clients to construct file relation graph for malware detection. The newly unknown malware can be detected by its association with the known files (benign or malicious). (See Section 3.1 for details).
- *Design enhanced Belief Propagation (EBP) algorithm for unknown file labeling:* Belief Propagation (BP) algorithm is a promising method for solving inference problems over graphs and it has also been successfully used in many domains (e.g., computer vision, coding theory) [150]. However, in our application, the algorithm should be greatly adapted, which is not a trivial process: we fine tune various components used in the algorithm and carefully design the message update and belief read-out functions for malware detection [31, 66] (See Section 3.2 for details).
- *Build effective active learning framework for malware detection:* Based on the constructed file-to-file relation graph, we design five graph-based features to represent each file and further analyze its relationship characteristics and have two significant findings: *Finding 1:* A file can greatly inherit the indirect influences from other files. *Finding 2:* (1) The importance of each file is different; (2) The neighbors of the important malware are associated through it, while the neighbors of the non-important malicious file are inclined to be a clique. We also use graph metrics to quantitatively validate these findings. Accordingly, we first apply Malicious Score Inference Algorithm (MSIA) to select the representative samples from the large unknown file collection for labeling, and then use EBP algorithm to detect malware [27] (See Section 3.3 for details).
- *Leverage Long Short-term Memory for graph representation learning:* To learn the representations of files in our constructed graph, we first generate file sequences based on the random walk, and then deploy Long Short-Term Memory (LSTM) for file sequence modeling and thus learn desirable file representations over graph, which will be fed to a Support Vector Machine (SVM) to train the classification model, based on which the unlabeled file can be predicted if they are malicious or not (See Section 3.4 for details).

- We explore the arms race between adversarial malware attack and defense to en-

hance the security of machine learning-based detection systems through analyzing adversarial attacks, and formulating secure-learning paradigms to counter the adversarial attacks. Our work on adversarial machinle learning in malware detection has resulted in 5 publications [32, 33, 28, 29, 30].

  - *Analyze adversarial attacks under different scenarios:* The attackers may have different levels of knowledge of the learning system [126]. We explore the adversarial attacks corresponding to the different scenarios, thoroughly assess the adversary behaviors through feature manipulations, adversarial cost, and attack goals, and accordingly present a general attack strategy for further investigations [32, 33, 29] (See Section 4.2 for details).
  - *Propose secure-learning paradigms against adversarial attacks:* Resting on the learning-based classifier which is degraded by the adversarial malware attacks, we propose three secure-learning models *SecDefender*, *SecureDroid*, and *DroidEye* to counter these attacks. In our proposed methods, *SecDefender* adopts classifier retraining technique on basis of our proposed adversarial attack model *AdvAttack* and enhances the robustness of the classifier using the security regularization terms; *SecureDroid* utilizes a novel feature selection method to build more secure classifier by enforcing attackers to increase the adversarial costs and maximize the manipulations, and introduces an ensemble learning approach to aggregate different individual classifiers constructed using our proposed feature selection method to improve system security while not compromising detection accuracy; *DroidEye* takes advantage of gradient masking for feature space, utilizes count featurization to transform the binary feature space into continuous probabilities encoding the distribution in each class (either benign or malicious) to reduce the adversarial gradient of the learning model, and then introduces softmax function (i.e., normalized exponential function) with adversarial parameter to find the best trade-off between security and accuracy for the classifier by tuning the adversarial parameter [32, 33, 28, 30] (See Section 4.3 for details).

- We develop practical systems integrating our proposed methods for comprehensive experimental studies on real sample collections from an anti-malware industry company. Specifically, we collect different sample sets from Comodo Cloud Security Center; based on these real sample collections, we construct practical systems based on our proposed methods and provide a series of comprehensive experiments

to empirically evaluate the performances of these methods [33, 28, 32, 31, 27, 30] (See Section 3.2.3, 3.3.3, 4.3.4, 4.4.4, and 4.5.3 for details).

In sum, by the date of this dissertation, we have had 9 publications [31, 27, 66, 143, 32, 33, 28, 29, 30] in intelligent malware detection using file-to-file relations and adversarial machine learning in malware detection, including the prestigious IEEE EISIC'2017 Best Paper Award.

1.4 Organization of This Dissertation

The rest of the dissertation is organized as follows. Chapter 2 first discusses the development of malware including its definition in Section 2.1.1, taxonomy in Section 2.1.2, and its history in Section 2.1.3; then specifies the development of malware detection techniques in Section 2.2. Chapter 3 describes file-to-file relation investigation and graph construction in Section 3.1, designs an enhanced Belief Propagation algorithm for unknown file labeling in Section 3.2, builds effective active learning framework for malware detection in Section 3.3, and leverages Long Short-term Memory for graph representation learning in Section 3.4. Chapter 4 presents the adversarial attacks under different scenarios in Section 4.2, and secure-learning paradigms *SecDefender* in Section 4.3, *SecureDroid* Section 4.4, and *DroidEye* in Section 4.5. Chapter 5 summarizes this dissertation in Section 5.1 and addresses the future work in Section 5.2.

Chapter 2

Development of Malware and Malware Detection Techniques

2.1 Development of Malware

2.1.1 Definition of Malware

Malware, short for ***mal***icious soft***ware***, generally refers to software programs that are designed to deliberately fulfill different harmful intents of an attacker [12, 51, 36], such as disturbing system operations, encrypting, stealing or deleting sensitive data, hijacking or altering core computing functions and monitoring computer activities of users without their permission, or even bringing down servers and critical infrastructures [106, 65]. Besides that, some other definitions have been also offered to describe malware [70]: Grimes defined malicious code as "any software program designed to move from computer to computer and network to network in order to intentionally modify computer systems without the consent of the owner or operator, that includes viruses, Trojan horses, worms, script attacks, and rogue Internet code" [56]; Vasudevan et al. described malware as a generic term that encompasses viruses, trojans, spywares and other intrusive code [123]; Thomas et al. also described malware as a general term used by computer professionals to mean a variety of forms of hostile, intrusive, or annoying software or program code [122]; while Saracino et al. considered that malware hides treacherous code performing actions in the background that threatens the user privacy, the device integrity, or even user's credit [113]. In a nutshell, the typical characteristics of malware can be depicted as destructive, unauthorized, stealthy, and transmissible, and they may infiltrate the systems through the vulnerable services over the network, the downloading process from

the Internet, or being tricked by the attackers into deliberately executing malicious codes on their machines [144, 46, 107].

Driven by economic benefits, malware industry has invented automated malware development toolkits (e.g., Zeus, Kronos, MPack, exploit kit) to produce and mutate hundreds of thousands of malicious codes per day using techniques, such as instruction virtualization, packing, polymorphism, emulation, and metamorphism [119]; these malware development toolkits also lead to a massive proliferation of new malware samples due to their wide availability [144]. As a result, malware has been rapidly gaining prevalence, spread and infected computing devices at an unprecedented rate around the world. According to AV-TEST Institute's statistics [4], as shown in Figure 2.1, over 350,000 new malicious programs and potentially unwanted applications are currently registered everyday, while more than 800 million malware have been created in the last ten years. This has posed serious and evolving security threats to Internet users.

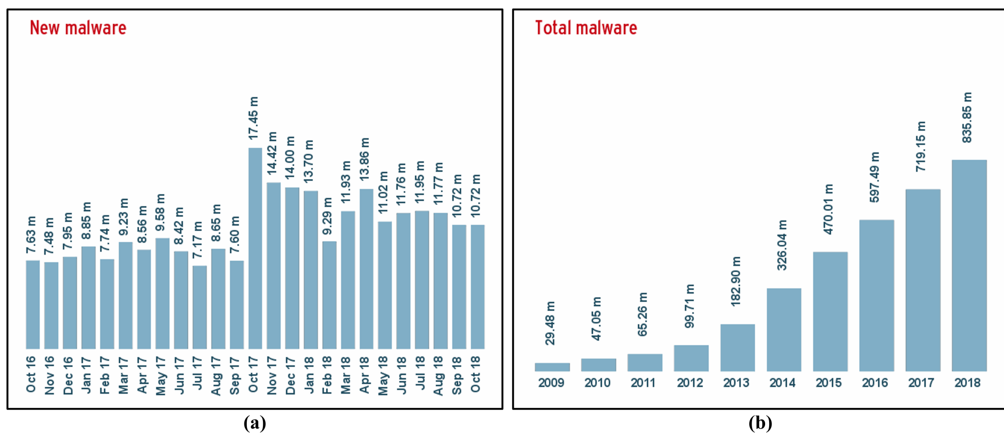


Figure 2.1: (a) New malware created in the last two years, and (b) total malware created in the last ten years [4].

2.1.2 Taxonomy of Malware

Malware comes in wide range of variations like virus, worm, backdoor, Trojan-horse, spyware, rootkit, adware, bot, scareware, ransomware [51], which are varying in different purposes and proliferation ways and containing unique characteristics and traits. We would like to provide a brief overview of these most common and prevalent types of malware as follows [144, 106, 70].

- *Virus:* A virus (e.g., Creeper, Elk Cloner) is a piece of code that replicates by inserting itself into other software programs, files, or the boot sector of the hard

drive. A program that a virus has inserted itself into is infected, and is referred to as the virus's host. Viruses cannot run independently since they need to be activated by their host programs.

- *Worm:* A worm (e.g., Love Gate, SQL Slammer) is a program that can run independently without a host program and propagate a fully working version of itself to other machines. Worms usually spread without any human interaction or directives from the malware authors.

- *Backdoor:* A backdoor (e.g., Sobig, Mydoom) is a malicious program that bypasses authentication procedures to access and thus compromise a system via a network. Additionally the use of a rootkit or code obfuscation makes the backdoors very difficult to locate.

- *Trojan horse:* A Trojan horse (e.g., Zeus, ZeroAccess) is a software program designed to appear as a legitimate program, thereby tricking a user into installing it onto their computing devices; when embedded by its designer in an application or system, the Trojan horses will perform malicious and unauthorized actions in the backend. Generally Trojans are responsible for the theft or destruction of data.

- *Spyware:* Spyware (e.g., keyloggers, web beacons) refers to a type of malware that has been designed to gather data and information about users and also observe their activity without users' knowledge and consent.

- *Rootkit:* A rootkit (e.g., Knark, Adore) is a form of malware that obtains administrator-level access to the victim's system. After the installation process, the program provides threat actors root or privileged access to the system. Rootkits can be used in user-mode or tamper with operating system structures as a device driver or a kernel module.

- *Adware:* Adware, or malvertising (e.g., Fireball, BaiduBarz), not only presents unwanted advertisements to the users to generate revenue, but also use authorized online advertising to spread malicious software.

- *Bot:* A bot (e.g., Agobot, Sdbot) is a piece of malware that allows the bot master to remotely control the infected system. Bots typically spread through exploiting software vulnerabilities or employing social engineering techniques to allure unsuspecting users to execute malware binaries. Once a system has been infected, the

bot master can transform these individual victimized systems into a vast network, called a botnet.

- *Scareware:* Scareware is a form of malware that utilizes social engineering to lure a user into buying and downloading unwanted software, such as fake antivirus software, which has posed severe financial and privacy-related threats to the victims.
- *Ransomware:* Ransomware is one of the most popular malware in recent years, which installs stealthily on a victim's computer and executes a cryptovirology attack that prevents users from accessing their system or personal files, and demands ransom payment from the victim in order to regain access.
- *Cryptocurrency mining malware:* Cryptocurrency mining malware (e.g., Coinhive), also simply called cryptojacking, refers to software programs and malware components developed to take over a computing device's resources and use them for cryptocurrency mining without a user's explicit permission [118].
- *Hybrid malware:* Hybrid malware combines two or more other forms of malware into a new type to achieve more powerful attack functionalities.

2.1.3 History of Malware

The history of malware starts back in 1949 when John von Neumann began working on self-reproducing automatons through "Theory and Organization of Complicated Automata" [125]; it seems no one attempted to implement these automatons to cause damage to the system. The root of malware came into life around 1970 named "Creeper", an experimental self-replicating program written by Bob Thomas that gained access via the ARPANET and copied itself to the remote system where the message "I'm the creeper, catch me if you can!" was displayed [34]. The term "Virus" was first coined by Fred Cohen in 1985 [88]. After that, a massive and outrageous malware industry was born [105]. Kingsoft reported that the average number of infected computers per day was between 2-5 million [76]; while cybercrime will cost the world $6 trillion annually by 2021, up from $3 trillion in 2015 [37]. In this section, we would like to take a brief look at the development of malware and get to know how it evolved and impacted the world as follows [105, 144, 52, 129].

- *1970–1979:* Creeper, most commonly recognized as the first computer virus, was created by Bob Thomas in 1971 as an experimental self-replicating program that

corrupted DEC PDP-10 computers running the TENEX operating system; the Wabbit (or rabbit) virus was created in 1974 that blocks up the system of a computer through multiple self-replicating; PERVADE virus appeared in 1975 that adheres to other programs and allows them to spread its copies; the term "Worm" was also introduced by John Brunner in 1975.

- *1980–1989:* Elk Cloner virus, found in Apple II systems, was written by high school student Richard Skrenta in 1981 that resulted in one of the earliest large-scale virus outbreak to affect personal computers; Brain virus was released in 1986 that is considered as the first virus to infect MS-DOS computers and the first IBM PC compatible virus, while in the same year PC-Writer Trojan was created as one of the earliest Trojans; Vienna virus appeared in 1987 that was regarded as the first virus to infect both COM files and EXE files; Christmas Tree virus was created in the same year that was the first widely disruptive replicating network virus; Morris Worm was released in 1988 to infect a substantial percentage of computers connected to ARPANET while its author (i.e., Robert Morris) became the first malware author convicted for his crimes; AIDS Trojan was spread in 1989 that encrypts all filenames on the system and asks for payment, which is considered as the first known ransomware.

- *1990–1999:* Chameleon viruses were developed by Mark Washburn in 1990 as the first family of polymorphic viruses; the first Macro virus, called Concept, was created in 1995 used to infect Microsoft Word documents; in 1996, the first virus designed specifically for Windows 95 files - Boza, the first Excel macro virus - Laroux, and the first Linux virus - Staog were released; in 1999, Melissa virus was the first mass-emailing virus that utilizes Outlook address books from infected machines, and sends the copy of itself to 50 people at a time, while ExploreZip worm was detected to destroy Microsoft Office documents.

- *2000–2009:* Starting from 2000, Internet and email worms were prevailing across the globe, and malware toolkits drove malware to grow significantly in its number and dissemination. In 2000, the ILOVEYOU worm, one of the most damaging worms ever, was created by a computer science student that infected millions of computers worldwide and costed more than $5.5 billion in damages; In 2001, different worms were detected that targeted Microsoft systems, such as Sadmind, Sircam, Code Red, Nimda, and Klez; SQL Slammer Worm, one of the fastest spreading

worms of all time, was created in 2003 that caused massive Internet access disruptions worldwide; Cabir Virus was released in 2004, which was widely acknowledged as the first mobile phone virus; In the same year, the first internet worm - Witty, and the first known webworm Santy were also launched; the first-ever malware for Mac OS X, a trojan-horse known as OSX/Leap-A or OSX/Oompa-A was announced in 2006; computer worm Conficker was found in 2008 causing some of the worst damage seen since SQL Slammer.

- *2010–present:* Since 2010, the sophistication of malware has been significantly evolving, that results in advanced malware with different evasion tactics. Stuxnet, a malicious computer worm, was detected in 2000, which was the first worm to attack SCADA systems and one of the most resource-intensive bits of malware created to date; Zeus Trojan, or Zbot released in 2011 was one of the most successful pieces of botnet software in the world, impacting millions of machines; Cryptolocker discovered in 2013 was one of many early ransomware programs; Cerber detected in 2016 was one of the heavy-hitters in the ransomware sphere, and one of the most prolific crypto-malware threats; WannaCry Ransomware and its variants have been spreading globally since 2017 by encrypting data and demanding ransom payments, which is one of the most prevalent and diabolical malware in recent years.

2.2 Development of Malware Detection Techniques

In order to combat the evolving malware attacks and protect legitimate users from these threats, the major defense is the software products from anti-malware companies. With more and more sophisticated malware samples emerging in the wild, both anti-malware industry and researchers have developed various countermeasures for malware detection. In the following sections, we briefly introduce the progress of intelligent malware detection.

2.2.1 Signature-based Malware Detection

Signature-based methods are widely used in anti-malware software products from different companies to provide the major defense against malware, such as Comodo, Kaspersky, Kingsoft, and Symantec [49, 48]. A signature is a short sequence of bytes, which is often unique to each known malware, allowing newly encountered files to be correctly identified with a small error rate [74, 92]. In addition to anti-malware software

products, some early research efforts have also been conducted on signature-based malware detection. Sun et al. [119] developed a signature based malware detection system called SAVE (Static Analyzer of Vicious Executables) which extracted the signatures from the original malware with the hypothesis that all versions of the same malware share a common core signature; Venugopal et al. [124] detailed a signature matching algorithm to scan malware in mobile devices. This method is traditionally known as time and labor consuming and less responsive to new threats as that signatures are often manually generated, updated, and disseminated by domain experts [144]. As introduced above, the malware industry has invented automated malware development toolkits to create and mutate hundreds of thousands of malicious codes per day which can easily slip through such traditional signature-based malware detection [149, 144, 61].

2.2.2 Heuristic-based Malware Detection

To address the aforementioned challenges, heuristic-based methods were proposed as complements to signature-based methods for malware detection [20]. As opposed to signature-based malware detection, which looks to match signatures found in files with that of a database of known malware, heuristic-based detection uses rules and/or patterns determined by experts to look for behaviors which may indicate malicious intent and thus discriminate malware samples and benign files [144]. These rules and patterns should be generic enough to be consistent with variants of the same malware threat, but not falsely matched on benign files [45]. Some classic heuristic-based detection techniques include [85, 24, 109]: neural networks(NNs) being adopted for their adaptability to environmental changes and their ability of prediction [89, 24]; fuzzy logic using approximation for logic rather than precise classical logic [91, 109]; genetic algorithm applying principles of evolutionary biology such as inheritance, mutation, selection and combination for deriving classification rules and selecting appropriate features or optimal parameters for malware detection [17, 91]. Heuristic-based malware detection is an effective way to identify unknown threats for the most up-to-date real-time protection, but there are downsides: (1) the analysis of malware samples and the construction of rules/patterns by domain experts is often error-prone, which results in high false positives; (2) this sort of scanning and analysis is time-consuming, which may slow-down system performance [144]. Considering that the speed of malware creation is faster than rules/pattern construction, the unknown files make the clients become more and more overloaded.

2.2.3 Machine Learning-based Malware Detection

To overcome the problem of the clients being heavy and keep the malware detection effective and efficient, cloud-based detection [149] has been recently used by most of the anti-malware vendors, the scheme of which can be summarized as "blocking invalid software programs from a blacklist and authenticating valid software programs from a white list at the client (user) side, and predicting any unknown files (i.e., the gray list) at the cloud (server) side and quickly producing the verdict results to the clients" [149, 144]. More specifically, the signature sets on the clients are first exploited to scan the newly received files; those files that cannot be recognized by existing signatures will be labeled as unknown files and their information will be collected and sent to the cloud server; the learning models constructed on the cloud side will classify the unknown files as malware or benign files, and send back the classification results to the client side immediately. Cloud-based detection enables an up-to-date security solution for malware detection [144]. However, the unknown files in the gray list is constantly increasing. According to the AV-TEST Institute's report, over 350,000 new malware are released everyday [4]. This calls for intelligent techniques to support efficient and effective malware detection on the cloud side.

Since the quantity, diversity and sophistication of malware have significantly increased in recent years, in order to effectively and efficiently detect malware from the real and large daily sample collection, new, intelligent malware detection systems have been developed by applying machine learning techniques [140, 7, 90, 139, 142, 95, 60, 39, 67, 68, 151, 131, 132, 137, 69]. In these methods, malware detection is a two-step process: feature extraction and classification/clustering. The performance of such malware detection methods critically depend on the extracted features and the categorization techniques. We provide a comprehensive investigation on machine learning-based malware detection techniques as follows.

- *Classification:* Naïve Bayes on the extracted strings and byte sequences was applied in [114, 50], which claimed that Naïve Bayes classifier performed better than traditional signature-based method. Kolter et al. [79] focused on static analysis of the executable files and compared Naïve Bayes, Support Vector Machine and Decision Tree based on the n-grams. Wang et al. [128] extracted registries and activity network from spyware and applied Support Vector Machine for surveillance spyware detection. Santos et al. first used n-grams, strings, and OpCode to build Decision Tree and used dynamically extracted behaviors to formulate kNN for malware

detection [111], and then they further proposed semi-supervised algorithms (i.e., collective classification models) on various features for unknown malware detection [112]. Ye et al. [139, 146, 148] proposed IMDS, Hierarchical associative classifier (HAC), and CIMDS performing associative classification on Windows API calls extracted from executable files. Shah et al. [116] applied various feature selection algorithms to obtain the feature sets from PE files and used Artificial Neural Networks to detect new and unknown malware. Kong et al. [80] extracted the function call graph from each program, collected various types of fine-grained features at the function level, and then applied an ensemble of weighted classifiers for malware detection. Cesare et al. [23] explored string similarity metrics for malware detection based on k-subgraphs and q-grams of structured control flow graphs, while Anderson et al. [2] used similarity metrics on instruction traces to differentiate malware and benign files.

- *Clustering:* Hou et al. [65] developed the intelligent malware detection system using cluster-oriented ensemble classifiers resting on the analysis of Windows API calls. Most of these existing researches are built on shallow learning architectures, which only made use of the files with class labels (either malicious or benign) during the training phase, while ignoring the important information from the large number of unlabeled file samples, which leave a large room for improvement. Bailey et al. [8] proposed a hierarchical clustering technique that describes malware behavior in terms of system state changes and automatically categorized these profiles of malware into groups that reflect similar classes of behaviors and demonstrated how behavior-based clustering provides a more direct and effective way of classifying and analyzing Internet malware. Ye et al. [145] presented an Automatic Malware Categorization System (AMCS) on function-based instruction sequences and instruction frequency that groups malware samples into families sharing some common characteristics using a cluster ensemble by aggregating the clustering solutions generated by different base clustering algorithms.
- *Deep learning:* Due to its superior ability in feature learning through multilayer deep architecture [62], deep learning is feasible to learn higher level concepts based on the local feature representations [98]. As a result, researchers have paid much attention to deep learning methods in the domains of malware detection [60, 84, 98, 71, 142]. Ouellette et al. [98] extracted control flow graphs to present malware samples, and used a deep probabilistic model (sum-product network) to

compare the similarities between the unknown file samples and those of representative sample features from known classes of malware. Jung et al. [71] used the features of header, tags, bytecode and API calls and utilized an ensemble learner consisting of different deep learning networks (e.g., deep feed-forward neural network, deep recurrent neural network) to classify the Adobe Flash file samples. Li et al. [84] proposed a hybrid malicious code detection approach on the basis of AutoEncoder and Deep Belief Network, where AutoEncoder was used to reduce the dimensionality of data, and a Deep Belief Network was applied to detect malicious code. Hardy et al. [60] and Ye et al. [142] both exploited API calls as inputs; the difference is that Hardy et al. developed Stacked AutoEncoders (SAEs) while Ye et al. formulated a heterogeneous deep learning framework composed of an AutoEncoder stacked up with multilayer restricted Boltzmann machines (RBMs) for malware detection.

- *File relations:* Besides those features stated above (e.g., strings and byte sequences, n-grams, API calls, function call graph, control flow graphs, etc.) extracted from file contents, file-to-machine relation graphs [25] and file-to-file relation graphs [149, 121] were also used as the features for malware detection. Chau et al. [25] and Tamersoy et al. [121] explored standard Belief Propagation (BP) algorithm to implement semi-supervised learning for malware detection. Ye et al. [149] proposed a semi-parametric classification model for combining file content and file relations together for malware detection. Karampatziakis et al. [72] built regression classifiers based on graphs induced by file relationships for malware detection. They showed that the system's detection accuracy could be significantly improved using the proposed method. File relation graphs have been starting to be leveraged to solve malware detection problems, but all these existing works merely take advantage of the graph structure while not going further to analyze the critical information about their properties and characteristics of the relationships among different file samples.

2.2.4 Adversarial Machine Learning in Malware Detection

Machine learning techniques offer unparalleled flexibility in automatic malware detection. However, machine learning itself can be a target of attack by a malicious adversary [87, 126, 14, 13, 127, 40, 11, 15]. In some cybersecurity domains, there are ample evidences that show adversaries can actively manipulate the data to evade the detection

[38, 152, 87, 18, 14]. For example, in the domain of spam email detection, Dalvi et al. [38] examined the cost for measuring each feature of the dataset using Naïve Bayes classifier, and proposed an optimal strategy for the adversary to play against the classifier. Zhang et al. [152] took gradient steps to find the closest evasion point $\mathbf{x}'$ to the malicious sample $\mathbf{x}$. The Adversarial Classifier Reverse Engineering (ACRE) framework [87] was introduced to study how an adversary can learn sufficient information to construct adversarial attacks using minimal adversarial cost. Brückner et al. [18] presented the interaction between the learner and the data generator as a static game, and explored the adversarial conditions and properties to find the equilibrial prediction model in the context of spam email filtering. All these adversarial attacks prompt increasing research efforts to improve the security of machine learning.

Specifically, the defense methods can be generally divided into four categories: Stackelberg game theories [19, 58, 18, 127], feature operations [152, 82, 53], retraining frameworks [54, 133, 83], and ensemble classifier systems [16, 40, 78]. To apply Stackelberg game theories, Bruckner et al. [18] first presented the interaction between the learner and the adversary as a static game, and explored the adversarial properties to find the equilibrium prediction model; they then further simulated the interaction as a Stackelberg competition, and derived an optimization problem to determine the solution of this game [19]. Wang et al. [127] modeled the adversary action as it controlling a vector $\boldsymbol{\alpha}$ to modify the training data set $\mathbf{X}$, and transformed the classifier into a convex optimization problem. More recently, feature operation methods have also been proposed to counter some kinds of adversarial data manipulations, such as feature deletion [53], feature clustering [82], feature reduction [152], etc. In addition, retraining frameworks are becoming more and more widely applied to boost the resilience of learning algorithms through: (1) adding adversarial samples to the training data that evade the previously computed classifier [83, 54], and (2) manipulating the training data distribution that its distribution is matched to the test data [133]. To improve the security of machine learning under generic settings, some research efforts have been devoted to multiple classifier systems. Kolcz et al. [78] applied averaging method resting on random subsets of reweighted features to produce a linear ensemble classifier. Biggio et al. [16] built a multiple-classifier system to improve the robustness of the classifier through bagging, and the random subspace method. Debarr et al. [40] explored randomization to generalize learning model by randomly choosing dataset or features, and estimated parameters that fit the data best. In these ensemble learning systems, randomization is the main method

for feature selection. Though these theories and approaches are promising, the application of adversarial machine learning into malware detection domain has been scarce with the exception that Šrndic et al. [126], Xu et al. [135], and Demontis et al. [41] all exploited PDF malware or Android malware as a case study to evaluate the security of learning-based classifiers (e.g., PDFrate, Hidost, and Drebin). With the popularity of machine learning based detections, such adversaries will become even more violent.

Chapter 3

Intelligent Malware Detection Utilizing File-to-file Relations

File-to-file relations, such as file-co-existence, can provide invaluable information in malware detection and make evasion harder [149, 25, 121, 72]. To better understand the properties of file-to-file relations (i.e., malware-malware, malware-benign, benign-benign relations), we'd like to take a further step to delve deeper into the relationship characteristics of malware and benign files. It is of interest to know:

- *How can we construct the file-to-file relation graph between malware and benign files?*
- *What graph-based features, relationship characteristics, and representations can be employed for malware detection?*
- *Instead of traditional machine learning-based classification methods, how can we build effective learning frameworks over graph for malware detection?*

More specifically, we analyze and utilize the relations among file samples to construct file relation graph. Resting on the constructed file-to-file relation graph, we first present our enhanced Belief Propagation (EBP) algorithm for malware detection; then, we design several new and robust graph-based features to represent each file and further investigate the relationship characteristics of relation graph, on the basis of which, we propose an active learning framework that applies Malicious Score Inference Algorithm (MSIA) to select the representative samples from the large unknown file collection for labeling and then uses EBP algorithm to detect malware; afterwards, we learn representations for files over graph using Long Short-term Memory (LSTM), which will be fed to SVM to train

the classification model and predict if the unlabeled files are malicious or not. To the best of our knowledge, this is the first work of investigating the relationship characteristics of the file-to-file relations in malware detection using social network analysis.

3.1 File-to-file Relation Graph Construction

In this section, we (1) first introduce the file relation graph construction, and (2) then provide deep analysis of malware's social relation network.

File Relation Graph Construction. Based on the collected data, we construct a file-to-file relation graph to describe the relations among file samples. Generally, two files are related if they are shared by many clients (or equivalently, file lists). The file relation graph is defined as $G = (V, E)$, where V is the set of file samples and E denotes the relations between file samples, which is shown in Figure 3.1.

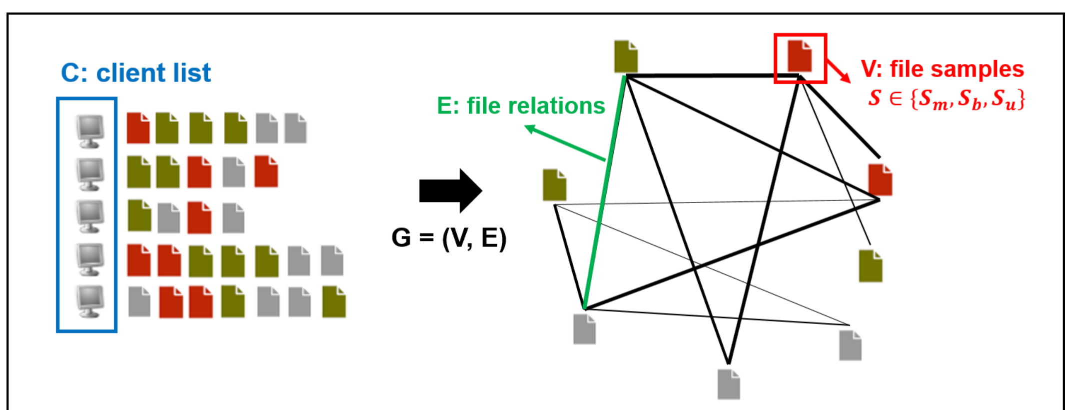


Figure 3.1: An example of file-to-file relation graph.

Given two file samples v_i and v_j, let C_i be the set of clients containing v_i and C_j be the set of clients containing v_j. $|.|$ represents the size of a set. The connectivity between v_i and v_j is computed as

$$con(v_i, v_j) = \frac{|C_i \bigcap C_j|}{|C_i \bigcup C_j|}. \tag{3.1}$$

If the connectivity between a pair of file samples is greater than the specified threshold (e.g., 0.5), then there is an edge between them. Each file is in a state $S \in \{s_m, s_b, s_u\}$ (s_m: malicious, s_b: benign, s_u: unknown). Assume that v_i is in state s_i and v_j is with state s_j, the weight of the edge between v_i and v_j which infers the probability that node

i and node j can be connected together is defined as

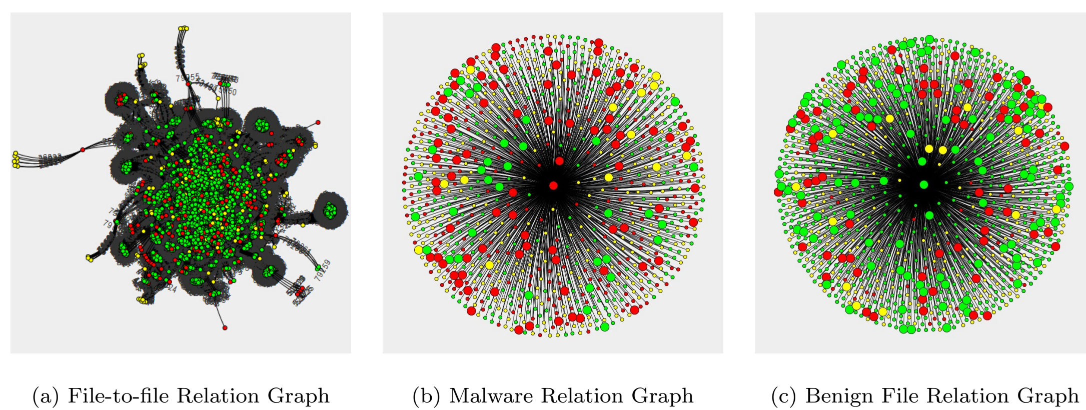
$$w(v_i, v_j) = \frac{|E_{s_i,s_j}|}{|E|}, \tag{3.2}$$

where $|E_{s_i,s_j}|$ is the number of the edges between all the files with states s_i and s_j, and $|E|$ is the number of all the edges. The weight of node v_i which denotes its popularity can be defined as

$$w(v_i) = \frac{|C_i|}{|C|}, \tag{3.3}$$

where C is the set of all the clients.

To visualize the file-to-file relation graph, we analyze the dataset obtained from Comodo Cloud Security Center, which contains the relationships between $60,724$ files (9,893 malware, 19,402 benign files and 31,429 unknown files) on $7,093$ clients. For the file relations collected from $7,093$ clients, we construct the graph consisting of $60,724$ nodes and $3,471,288$ edges. Figure 3.2(a) shows a part of the constructed graph, while Figure 3.2(b) and (c) give examples of a malware relation graph and a benign file relation graph with one-hop information respectively.



(a) File-to-file Relation Graph (b) Malware Relation Graph (c) Benign File Relation Graph

Figure 3.2: Visualization of file-to-file relation graphs: (a) a part of the constructed graph; (b) an example of a malware relation graph with one-hop information; (c) an example of a benign file relation graph with one-hop information (Red nodes denote malware, green nodes represent benign file, and yellow nodes are unknown file) [27].

Graph Property Overview. To gain an overview about the property of file relation graph, we used a subset of our data collection includes the file lists from 1000 clients which describe file co-existence relations between 1,540 malware, 7,687 benign files, and 2,250 unknown files. Figure 3.3 shows a zoom-in view of a part of the constructed file

relation graph. From Figure 3.3, we can see that many of the red nodes are associated with other red nodes and form some clusters, while the green nodes are also related to other green nodes and form their clusters. The nodes within the same cluster have strong relations with each other: (1) the red clusters may be the variants of malware families (e.g., family of online-game trojans); (2) the green clusters may be the related files of same applications (e.g., Acrobat installation archive and its related files).

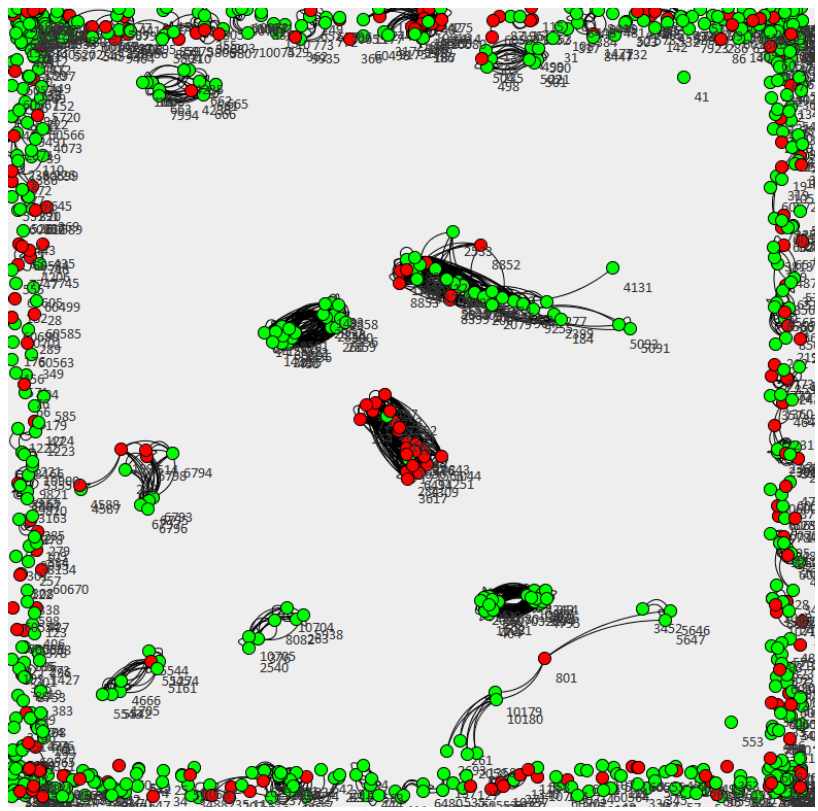


Figure 3.3: A zoom-in view of a part of the constructed file relation graph [66].

Based on the dataset described above, we also use fourteen measures in Table 3.1 to see the differences between benign file relation graph, ordinary malware (i.e., 1,220 malware whose the existence frequency is < 100) file relation graph and popular malware (i.e., 320 malware whose existence frequency is ≥ 100) file relation graph. In Table 3.1, from the comparisons of *G1* and *G2*, we can see that the measures of *components*, *component ratio*, *connectedness* and *fragmentation* are different between benign file relation graph and ordinary malware file relation graph; while from the comparisons of *G2* and *G3*, we can see that the measures of *avg degree*, *centralization* and *density* are different between ordinary malware file relation graph and top popular malware file relation graph. The different properties between benign file relation graph and malware

file relation graph enable us to discriminate malware and benign files, while the different properties between ordinary malware file relation graph and popular malware file relation graph may allow us to predict the trend of malware prevalence.

Table 3.1: Graph property comparisons

NO.	Measures	G1	G2	G3
1	H-Index	129	125	125
2	Avg Degree	49.842	**40.677**	**12.098**
3	Centralization	0.340	**0.344**	**0.081**
4	Density	0.018	**0.014**	**0.001**
5	Components	**103**	**11**	2
6	Component Ratio	**0.036**	**0.004**	0.000
7	Connectedness	**0.964**	**0.996**	1.000
8	Fragmentation	**0.036**	**0.004**	0.000
9	Closure	0.081	0.091	0.047
10	Avg Distance	2.744	3.056	3.408
11	SD Distance	0.631	0.836	0.717
12	Diameter	5	7	4
13	Breadth	0.625	0.645	0.689
14	Compactness	0.375	0.355	0.311

"G1": graph constructed based on 7,687 benign files and files co-exist with them, "G2": graph constructed based on 1,220 ordinary malware and files co-exist with them, "G3": graph constructed based on 320 popular malware and files co-exist with them.

3.2 An Enhanced Belief Propagation Algorithm for Malware Detection

In this section, we first introduce the preliminaries of BP, and analyze the reason why the standard BP fails for our application, then propose an enhanced BP for malware detection based on our constructed file relation graphs: we fine tune various components used in the algorithm and well design the message update and belief read-out functions for malware detection.

3.2.1 Standard Belief Propagation

Belief Propagation (BP) is a promising method for solving inference problems over graphs and it has also been successfully used in many domains (e.g., computer vision, coding theory) [150]. It was first proposed by Judea Pearl [102] to calculate marginal distribution in Markov Random Fields and Bayes Nets. Nodes of the graph perform as a local summation operation by iterations using the prior knowledge from their neighbors and then pass the information to all the neighbors in the form of messages [97]. By definition, the message is the neighbor node's opinion for the current node's probability of being in the designated status. The passing operation should cover every pair of connected nodes.

The key idea of BP is to update each node's message until the sum of messages converge or the iterations reach the designated number. Once the final messages are defined, the belief value of each node will be read out from all its neighbor nodes. The belief is the final result employed for inference. Figure 3.4 illustrates the message update of node j from its neighbor node i considering all the messages flowing into node i (except message from node j).

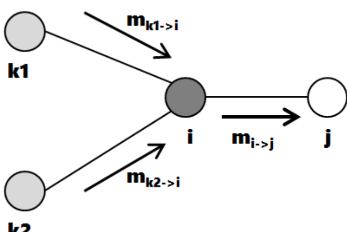


Figure 3.4: Message update from node i to node j

Mathematically, the message update equation in standard BP is

$$m_{i\to j}(v_j) = \sum_{v_i \in S} f_{i\to j}(v_i, v_j) g_i(v_i) \prod_{k\in N(i)/j} m_{k\to i}(v_i) \tag{3.4}$$

where $m_{i\to j}(v_j)$ is the message sent from node i to node j, node i's belief that node j is in the state v_j; both $g_i(v_i)$ and $f_{i\to j}(v_i, v_j)$ are typically called as energy functions, in which, $g_i(v_i)$ is the node potential, meaning the prior probability of node i being in the state v_i, while $f_{i\to j}(v_i, v_j)$ is the edge potential, referring the probability of node i being in the state v_i and node j being in the state v_j; S is the set of states; $N(i)/j$ is the set of nodes neighboring node i (not including node j). BP algorithm stops when message updates converge or a maximum number of iterations has finished. Then we calculate

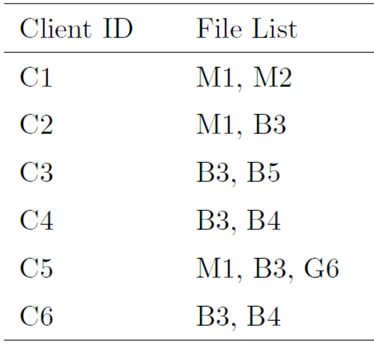
the belief value of each node as follow

$$b_i(v_i) = g_i(v_i) \prod_{k \in N(i)} m_{k \to i}(v_i) \tag{3.5}$$

In general, we always normalize both message $m_{i \to j}(v_i)$ and belief $b_i(v_i)$, preventing values from underflow or overflow. The standard BP is commonly called *sum-product* (from its message-update equation). A simple variant, called *max-sum*, is used to estimate the state configuration with maximum probability.

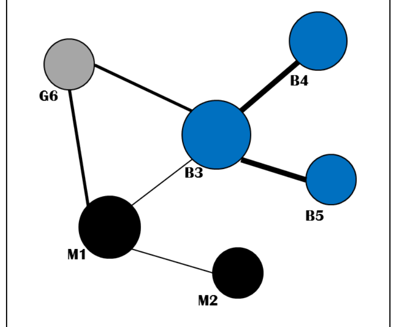
3.2.2 Enhanced Belief Propagation

To tailor BP algorithm to our problem, the energy function designs as well as message update and belief read-out are the key points. Unfortunately, the energy function designs based on the standard BP in AESOP [121] fail in our application. To put this into perspective, we use the example in Figure 3.5 for further illustration, in which "M" denotes malware, "B" denotes benign file, and "G" is unknown file. Figure 3.5(b) is the constructed file relation graph based on the sample dataset (note that the weights of the nodes and edges are different).



Client ID	File List
C1	M1, M2
C2	M1, B3
C3	B3, B5
C4	B3, B4
C5	M1, B3, G6
C6	B3, B4

(a) A sample datase



(b) file relation graph

Figure 3.5: A sample dataset and its file relation graph constructed

Table 3.2: The edge potential design in AESOP[121]

$f_{i \to j}(x_i, x_j)$	x_i: Malicious	x_j: Benign
x_i: Malicious	0.99	0.01
x_j: Benign	0.01	0.99

We employ the same energy functions designed in AESOP [121]: (1) the prior probability is 0.99 when the file is benign, 0.01 when the file is malicious, and 0.5 when the

file is unknown; (2) the edge potential design is shown in Table 3.2. When the message updates converge (within threshold 10^{-3}), the belief values of the data nodes (i.e., BP_Belief) are shown in Table 3.3. From the results (i.e., BP_Class) in Table 3.3, we can see that file $B3$ and file $G6$ are misclassified.

Table 3.3: The results of standard BP and EBP based on Figure 3.5

Nodes	BP_Belief	BP_Class	EBP_Belief	EBP_Class
M1	0.000000	M	0.022937	M
M2	0.260410	M	0.046170	M
B3	0.000000	M	0.135394	**B**
B4	0.645915	B	0.381034	B
B5	0.489330	B	0.317528	B
G6	0.169901	M	0.157791	**B**

In order to solve the problem above and make BP tailor to our application, we fine tune various components in BP and carefully design the message update and belief readout functions. Before doing that, we first analyze the meaning of each energy function in our case for malware detection. In Equation 3.4, $m_{i\to j}(v_j)$, $f_{i\to j}(v_i, v_j)$, and $g_i(v_i)$ represent message from node i to node j, edge potential, and node potential respectively. For malware detection problem, accordingly, $m_{i\to j}(v_j)$ means the probability of node i believes that the neighbor node j being a benign file; $f_{i\to j}(v_i, v_j)$ is the probability that node i and node j can be connected together; and $g_i(v_i)$ is the prior probability of node i being a benign file.

As described in Equation 3.2, the weight of edge between a pair of nodes is the probability of node i being in the state v_i and node j being in the state v_j, which is the edge potential $f_{i\to j}(v_i, v_j)$ in BP. Therefore, we use the weight of edge $w(v_i, v_j)$ (defined in Equation 3.2) between node i and j as the edge potential in our malware detection application, which is further illustrated in Table 3.4.

Table 3.4: The edge potential design in enhanced BP

$f_{i\to j}(x_i, x_j)$	x_i: Malicious	x_j: Benign
x_i: Malicious	$\lvert E_{s_m,s_m}\rvert \,/\, \lvert E\rvert$	$\lvert E_{s_m,s_b}\rvert \,/\, \lvert E\rvert$
x_j: Benign	$\lvert E_{s_b,s_m}\rvert \,/\, \lvert E\rvert$	$\lvert E_{s_b,s_b}\rvert \,/\, \lvert E\rvert$

For node potential, $g_i(v_i)$ is the prior probability of node i being a benign file. We

consider both its state and weight. Equation 3.6 shows our design of node potential in malware detection problem.

$$g_i(v_i) = \begin{cases} 0.5 + 0.5 * w(v_i) & \text{if } state(v_i) = s_b \\ 0.5 & \text{if } state(x_i) = s_g \\ 0.5 - 0.5 * w(v_i) & \text{if } state(v_i) = s_m, \end{cases} \tag{3.6}$$

where $w(x_i)$ is the weight of node i which can be calculated by Equation 3.3.

Instead of using sum-product, we redesign the message update equation as below:

$$m_{i \to j}(v_j) = \frac{1}{\beta} \sum_{v_i \in S} f_{i \to j}(v_i, v_j) g_i(v_i) \frac{\sum_{k \in N(i)/j} m_{k \to i}(v_i)}{p}, \tag{3.7}$$

where p equals to the number of the neighbors of node i (excluded node j) and β is a normalizing constant. In our application, we also initialize all the messages to 1. The belief read-out equation is designed as follow

$$b_i(v_i) = \frac{1}{\gamma} g_i(v_i) \frac{\sum_{k \in N(i)} m_{k \to i}(v_i)}{p}, \tag{3.8}$$

where γ is an adjustable constant.

Based on our enhanced BP (EBP) with the new energy functions as well as fine tuned message update and belief read-out equations above, using the same sample dataset in Figure 3.5, the belief value of each node (i.e., EBP_Belief) is shown in Table 3.3. From the results (i.e., EBP_Class) in Table 3.3, we can see that file $B3$ and file $G6$ are correctly identified as benign files; our adjusted BP algorithm performs well in malware detection problem. The implementation of EBP is given in Algorithm 1.

3.2.3 Experimental Results and Analysis

In this section, we conduct three sets of experiments to empirically evaluate our proposed EBP: (1) In the first set of experiments, we evaluate the effectiveness of our proposed EBP based on the file relation graphs for malware detection by comparing it with BP with sum-product, max-sum and AESOP in [121]. (2) In the second set of experiments, we evaluate our proposed algorithm compared with SVM and Decision Tree. (3) In the last set of experiments, we evaluate our proposed EBP algorithm in real industry application for malware detection.

Experimental Setup

We measure the malware detection performance of different methods using the evaluation measures shown in Table 3.5. All the experiments are conducted under the

Algorithm 1: ***EBP*** - An enhanced Belief Propagation algorithm based on file relation graphs for malware detection

Input: $G = (V, E)$: undirected weighted file relation graph(s), $w(x_i, x_j)$: the edge weights for file pairs x_i and x_j, $w(v_i)$: the node weights for files v_i

Output: class label of each file

Initialize (file states, messages): 0.99 when the file is benign, 0.01 when the file is malicious, and 0.5 when the file is unknown;
Calculate node potential $g_i(v_i)$ for each file v_i;
Calculate edge potential $f_{i \to j}(x_i, x_j)$ for each pair of associated files x_i and x_j;
while *messages haven't converged or iteration hasn't reached* **do**
    for *each file in graph(s)* **do**
        Message Update: $m_{new} \leftarrow m_{old}$;
    end
    Normalization;
end
Calculate belief value of each file using its neighbors' messages;
Define the threshold using the training data set;
Infer the status of each file according to the probability: benign when the probability is greater than the threshold; otherwise, malware

environment of 64 Bit Windows 7 operating system with 4th Generation Intel Core i7 Processor (Quad Core, 8MB Cache, up to 4.0GHz w/ Turbo Boost) plus 16G of RAM using Apache Pig, MySQL and C++.

Comparisons of Different Belief Propagation Algorithms

In this section, we conduct the experiments to evaluate our proposed EBP for malware detection based on the first dataset containing 4,675 files: 260 are malware, 2,583 are benign files and 1,832 are unknown (with the analysis by human experts, 1,627 of them are marked as benign and 14 are malicious). We also compare our proposed algorithm with standard BP in sum-product, max-sum, and AESOP in [121]. The results in Figure 3.6 show that our proposed EBP algorithm obtains the highest TPR and the lowest FPR that result in the best F1 measure and ACC, performing better than other

Table 3.5: The evaluation measures of malware detection performance

Measures	Specification
TP	Number of files correctly classified as malicious
TN	Number of files correctly classified as benign
FP	Number of files mistakenly classified as malicious
FN	Number of files mistakenly classified as benign
FPR	$FP/(FP + TN)$
$Precision$	$TP/(TP + FP)$
$Recall/TPR$	$TP/(TP + FN)$
ACC	$(TP + TN)/(TP + TN + FP + FN)$
$F1$	$2 \times Precision \times Recall/(Precision + Recall)$

three, due to our well designed energy functions and tuned message update as well as belief read-out.

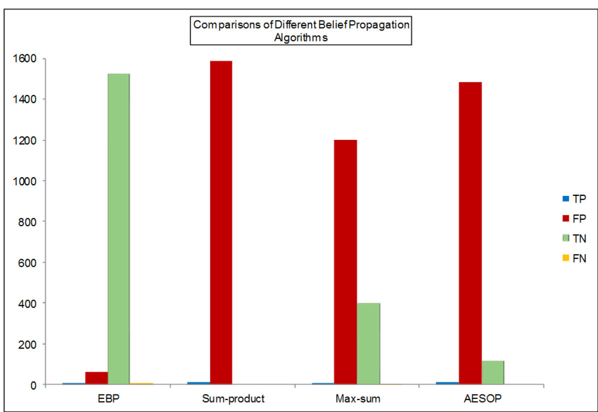
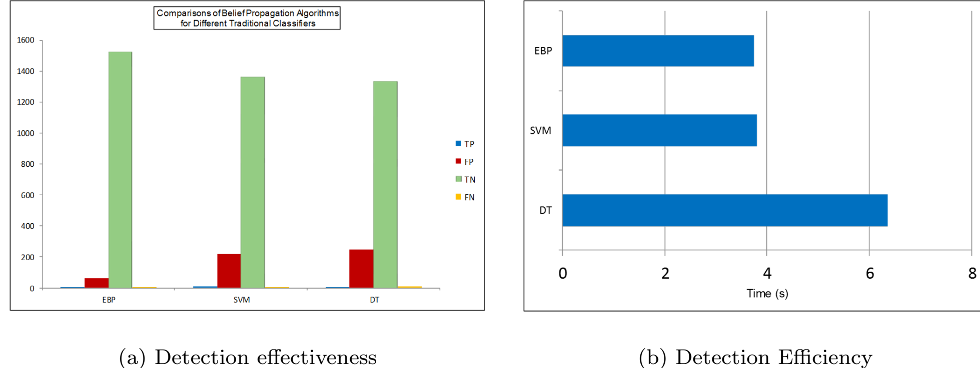


Figure 3.6: Comparisons of different belief propagation algorithms

Comparisons of Enhanced Belief Propagation Algorithm with Other Classification Approaches

In this section, we compare the malware detection effectiveness and efficiency of our proposed EBP algorithm and other classification approaches (Support Vector Machine (SVM) and Decision Tree (DT)) based on the same dataset in the previous section. Figure 3.7(a) show that the our proposed EBP algorithm outperforms the other two classifiers in malware detection effectiveness with the highest TPR and the lowest FPR. Figure 3.7(b) also shows that the EBP performs better than the other two classifiers in

malware detection efficiency with detection time of less than 4 second. The computation complexity of the EBP is $O(n^2)$, where n is the number of the file samples.



(a) Detection effectiveness (b) Detection Efficiency

Figure 3.7: Comparisons of malware detection effectiveness and efficiency between EBP Algorithm and other classification approaches.

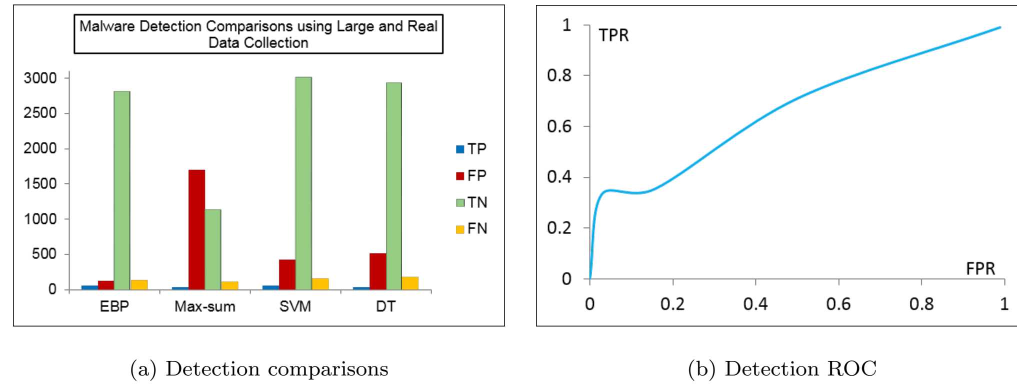
Enhanced Belief Propagation Algorithm Applied in Real Industry

In this section, we further evaluate the detection performance of our proposed EBP based on the large and real data collection from Comodo Cloud Security Center that includes 69,165 files: 2,883 malware, 19,142 benign files, and 47,140 unknown files. Here, we use the file labels of the unknown files available two weeks later with the analysis by the anti-malware experts of Comodo Security Lab for evaluation. 3,653 of the unknown files are labeled manually: 212 are malware and 3,441 are benign files. Since the sum-product and AESOP in [121] completely fail in our case, we compare our proposed EBP with max-sum BP, SVM and DT in this section. The results in Table 3.6, Figure 3.8 demonstrate that our proposed EBP algorithm outperforms others in malware detection based on the large and real data collection.

Table 3.6: Malare detection comparisons using large and real data collection

Predicting	TP	FP	TN	FN	ACC
EBP	51	119	2,803	130	0.9197
Max-sum	36	1,698	1,137	117	0.3926
SVM	59	429	3,012	153	0.8407
DT	38	512	2,929	174	0.8019

Remark: we use threshold gap to remove some indistinguishable samples, so TP+FN, FP+TN are not equal to 212, 3441; these numbers vary in different algorithms either



(a) Detection comparisons

(b) Detection ROC

Figure 3.8: Malare detection comparisons using large and real data collection.

3.3 Active Learning in Malware Detection

In this section, we further gain deeper insight into the file relation graph, which includes designing its graph-based features, and revealing its relationship characteristics, and then based on our findings, we propose an active learning framework for malware detection that applies MSIA to select the representative samples from the unknown files for labeling and then uses EBP to detect the remaining malware.

3.3.1 Gaining Insight into the Semantic Relatedness

Designing Graph-based Features

To counter malware's evasion tactics, after the construction of the file-to-file relation graph, we further investigate several robust graph-based features for malware detection. Ideal features are either difficult or costly to evade, even when malware is obfuscated. In this section, on the basis of special characteristics of the file relationships between malware and benign files, we design five robust and representative graph-based features for malware detection, which are described in details in the followings.

Vertex degree. The degree of a vertex in a graph is the number of edges incident to the vertex, which can specifically represent the association between the vertex and its neighbors [42]. In the file relation graph, we use the degree of malware (DoM) and degree of benign files (DoB) to capture the association between the file and its neighbors. These two metrics can be calculated as

$$DoM(v) = |\delta_m^v|, \;\; DoB(v) = |\delta_b^v|, \tag{3.9}$$

where $|\delta_m^v|$ is the total number of vertex v's malicious neighbors, and $|\delta_b^v|$ is the total number of vertex v's benign neighbors. As the moral says that " man is known by the company he keeps", it's easy to understand that malware is more likely to have a larger DoM than DoB, and vice versa. To further support this point, we calculate the degree for each file in the collected dataset described above: 53.75% of malware have larger DoM than DoB; while only 3.10% of benign files have larger DoM than DoB.

Influence coefficient. For spammer detection, in [26], the authors used reposting and commenting coefficients to indicate the ability that a user affects others to repost or comment. In malware detection, we define the influence coefficient of malware and benign files by Equation 3.10 and Equation 3.11.

$$IoM(v) = \frac{\sum_{i=1}^{N} \log(Malware_Count(v_i) + 1)}{N}, \tag{3.10}$$

$$IoB(v) = \frac{\sum_{i=1}^{N} \log(Benign_Count(v_i) + 1)}{N}, \tag{3.11}$$

where N denotes the number of vertex v's neighbors and v_i denotes the ith neighbor of v. $Malware_Count(v_i)$ and $Benign_Count(v_i)$ represent the number of the malware and benign files directly connected to v_i respectively. A file can directly or indirectly inherit the goodness or malice from other files. Compared with vertex degree, which considers the information one-hop away from the node, the feature of influence coefficient takes the indirect influence from other files into consideration.

Local clustering coefficient. The local clustering coefficient of a vertex in a graph specifies how close vertices in its neighborhood are to being a clique [138]. For each vertex in the constructed file relation graph, its local clustering coefficient can be calculated as [138]

$$LCC(v) = \frac{2|e^v|}{k_v(k_v - 1)}, \tag{3.12}$$

where $|e^v|$ is the total number of edges built by all v's neighbors, and k_v is the degree of the vertex v. For benign files, different users may install different sets of applications according to their occupations, ages, etc. And these applications are unnecessary to have associations with each other. However, for malware, just specified groups of users would be infected by malware. When infected, not only one malicious software would appear in the client, but also its related files would be released or downloaded. For example, variants of trojans will always come together with trojan-downloader and co-exist in the clients. Therefore, malware will have a larger local clustering coefficient than benign files. To quantitatively validate this, we calculate the local clustering coefficient for each

file in the collected dataset described above: the average LLC for malware is 0.9387, while the average LLC for benign files is 0.7573.

Degree centrality. Degree centrality of a vertex is determined by the number of vertices adjacent to it. The larger the degree, the more important the vertex is [115]. In malware detection, degree centrality can be used to quantify the importance of a file, which can be computed as [115]

$$DC(v) = \frac{\delta(v)}{n-1}, \tag{3.13}$$

where $\delta(v)$ is the degree of the vertex v, and n is the number of vertices in the graph.

Closeness centrality. Closeness centrality measures the significance of vertices by quantifying their centrality. Central vertices tend to reach the whole graph more quickly than non-central vertices [115]. Closeness centrality factors in how close a vertex is to other vertices, which is computed as [115]

$$CC(v) = \frac{1}{n-1} \sum_{u \neq v}^{n} g(u, v), \tag{3.14}$$

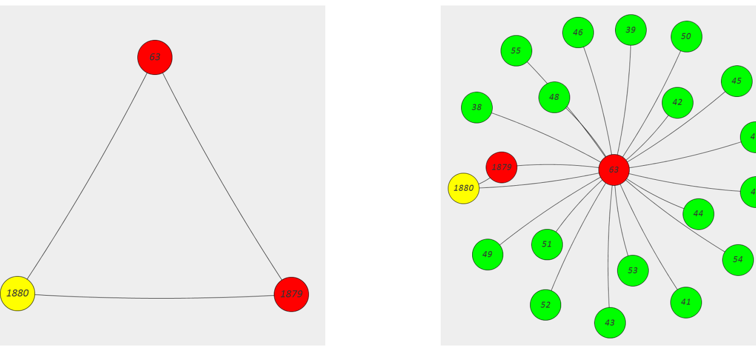
where $g(u, v)$ is the distance between the vertex u and vertex v, and n is the number of vertices in the graph. Malware attackers always use a shotgun approach to find victims and allure them to download variants of malicious files (e.g., trojans, adware). These files in the victim clients are always connected through the downloaders. Thus, the closeness centrality of those downloaders will be high.

Characterization of the Semantic Relatedness

After visualizing the constructed file-to-file relations and designing the graph-based features, we further analyze its relationship characteristics, and give the following observations.

Finding 1: A file can greatly inherit the indirect influences from other files in the file-to-file relations. Again, as the moral says "man is known by the company he keeps", in malware detection, a file's goodness or malice can be judged by the other files that always co-exist with it in the clients. However, sometimes, a file can not only be directly influenced by its neighbors, but also greatly inherit the influences from other files (e.g., its neighbors' neighbors). Figure 3.9 shows an example that the indirect influences is superior than the direct influences for file 1880 (marked in yellow node). To quantitatively validate this finding, we use the features of influence of benign files

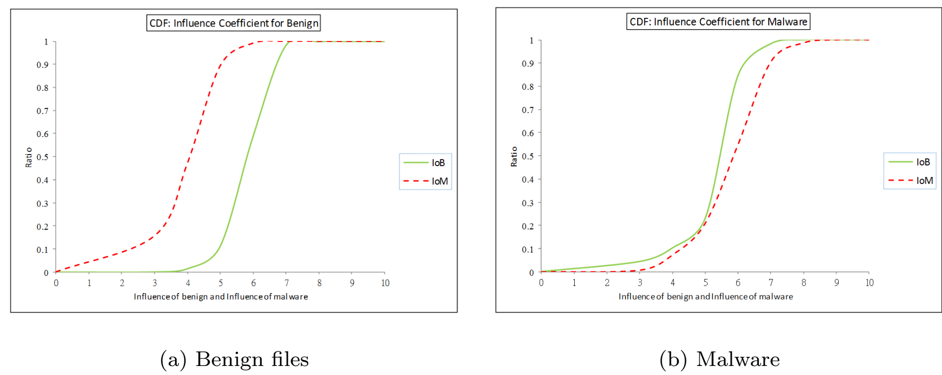
(IoB) and influence of malware (IoM) designed in the above section for measure. For file 1880, its IoB is 1.6290, while its IoM is just 0.6931, which means this file is more likely to be influenced by benign files, even though all the files it directly connects with are malware.



(a) The direct influences from its neighbors (b) The indirect influences from other files

Figure 3.9: Indirect influences superior than direct influences for file 1880 (yellow node)

To further illustrate, based on the collected dataset described in Section 3.1, we measure the indirect influences from other files for each node. Figure 3.10 displays the Cumulative Distribution Function (CDF) of IoB and IoM for both malware and benign files, which shows that both benign files and malware can greatly inherit the goodness and malice indirectly from other files.



(a) Benign files (b) Malware

Figure 3.10: The comparison of benign files and malware in IoB and IoM measures

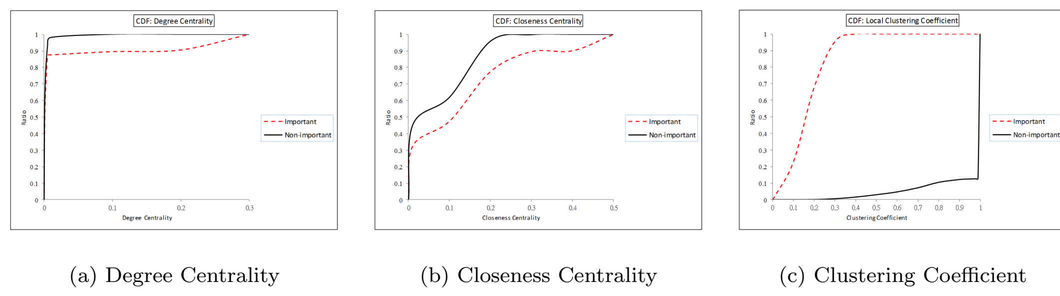
Possible Factor: To disseminate the malicious files, it is not uncommon for malware to be packaged into a software product (especially when it is free and open source) by the attackers. This would cause such kind of benign software to be closely related to malicious files, however, their neighbors of neighbors would not necessarily be. On

the other hand, variants of online game trojans may have indirect associations through the same kind of online game applications, since they target on stealing specific kind of online game accounts' information, but they are unnecessary to co-exist in the same clients.

From the observation above, we can see that a file's goodness or malice not only depends on its neighbors, but also greatly inherit the indirect influences from other files (e.g., its neighbors' neighbors). Furthermore, we are also interested to know: *(1) Is each malware of equal importance? (2) If not, what are the differences between the important malware and non-important ones?*

Finding 2: In the file-to-file relations, (1) the importance of each file is different; (2) the neighbors of the important malware are associated through it, while the neighbors of the non-important malicious file are inclined to be a clique.

To initially evaluate the importance of each node, we use degree centrality for measure (i.e., the importance is to evaluate if the file has high degree in the constructed file relation graph). Based on the collected dataset described in Section 3.1, we calculate the degree centrality of each file: about 2% of the malware have the degree centrality over 0.01, which are 10 − 1000 times larger than the remaining 98% ones. From this analysis, we can see that the importance of each malware is different: the larger the degree, the more important the vertex is[115]. Note that there is another interesting observation that those malware with larger degrees also have higher node weight values in the graph, which means the "important" malware are always with higher popularity. We mark those 2% malicious files with higher degree centrality as "important" malware, compared with the remaining 98% ones. Figure 3.11 (a) displays the CDF of degree centrality for the important malware and non-important ones.



(a) Degree Centrality (b) Closeness Centrality (c) Clustering Coefficient

Figure 3.11: The comparisons of "important" malware and "non-important" ones.

To further analyze the different characteristics of the important and non-important malware, we take insights to their graph structures. Figure 3.12 (a) illustrates an example of the relationship between an important malware A and its neighbors, while Figure 3.12 (b) shows the relations between its neighbors. From Figure 3.12 (a) and (c), we can see that both important and non-important malicious nodes with one-hop information have the star-structures, but the degree centralities of them are different. From Figure 3.12 (b) and (d), we can see that, the neighbors of the important malware are associated through it (the closeness centrality of the important malware A is 0.25), while the neighbors of the non-important malicious file are inclined to be a clique (the local clustering coefficient LLC of it is equal to 1). Figure 3.11 (b) and (c) display the CDF of local clustering coefficient and closeness centrality for the important malware and non-important ones respectively, which also validate the *Finding 2*.

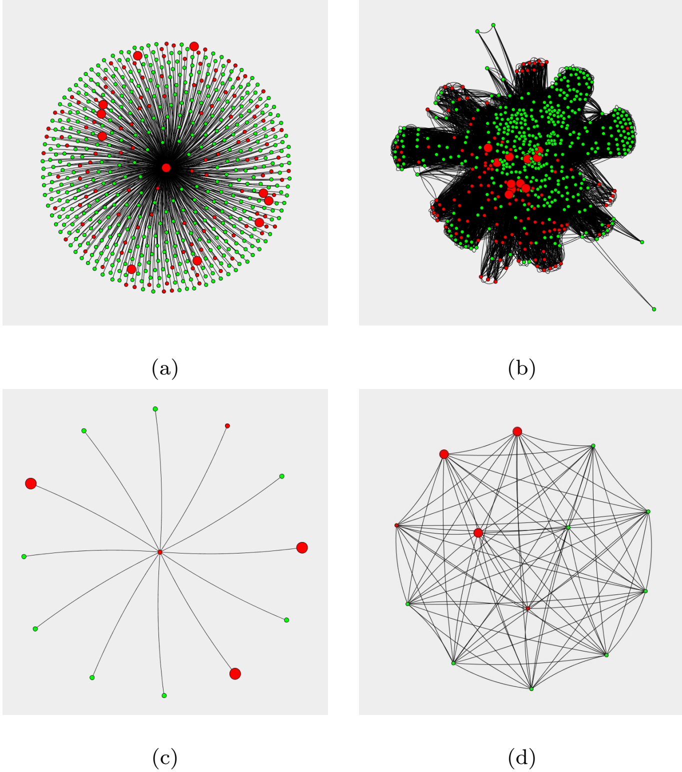


Figure 3.12: Graph structure comparisons of “important” and “non-important” malware. (a) Important malware A and its neighbors; (b) Relations between A’s neighbors; (c) Non-important malware B and its neighbors; (d) Relations between B’s neighbors [27].

Possible Factor: The importance of each malware is different, since the impacts different malicious files play are different. For example, a popular trojan or adware

downloader will infect more clients, compared with the specific kind of trojan or adware variants. The files co-exist with the popular downloader in different clients are unnecessary to have a close relationship among them, but are associated through the downloader; while the files co-exist with the variants of same trojan or adware are prone to be a clique, since they tend to be the same or similar kind of applications those trojans or adware target on.

3.3.2 Active Learning Framework

Via empirical analysis for the file-to-file relations, each node v_i (i.e., a file sample) in the constructed graph can be represented by its relations with other nodes and its graph-based features designed in Section 3.3.1, denoted as $Fv_i = \langle Rv_i, Gv_i \rangle$. Rv_i can be defined as

$$Rv_i = \langle v_{1i}, v_{2i}, ..., v_{ni} \rangle, \tag{3.15}$$

where $v_{ji} = \{0, 1\}$ (i.e., if $(v_j, v_i) \in E$, $v_{ji} = 1$; otherwise, $v_{ji} = 0$). Gv_i can be defined as

$$Gv_i = \langle DoM(v_i), DoB(v_i), IoM(v_i), IoB(v_i), LCC(v_i), DC(v_i), CC(v_i) \rangle. \tag{3.16}$$

Representative sample selection from the unknown file collection. To leverage the feedback from domain experts and thus to further improve the detection accuracy, selecting representative sample(s) from large unknown file collection for labeling is very important. For example, before being detected, the newly released Trojan-Downloader and its related trojans are collected from the user clients and may be marked as unknown. If we can recognize the Trojan-Downloader and have it labeled, then based on the constructed file relation graph, using the graph inference algorithm (e.g., EBP proposed above), its related trojans could be easily detected.

Active learning, as an effective paradigm to address the data scarcity problem, optimize the learning benefit from domain experts' feedback, and reduce the cost of acquiring labeled examples for supervised learning, has been intensively studied in recent years [93, 94]. In particular, with the abundance of graph and networked data in various application areas, active learning on graphs has received a lot of research attention [22]. For graph node classification, a general and powerful assumption is that connected nodes tend to be clustered into the same category, which means they will have the same class label. This assumption motivated the development of many classification techniques in

real-world networked data. In our application, *Finding 2* demonstrates that the importance of each file is different and the neighbors of the important malware are associated through them; therefore, selecting those important and representative malware from the large unknown file collection for labeling is significantly reasonable to further improve the detection performance.

In spammer detection, Yang et al. [136] proposed a Malicious Relevance Score Propagation Algorithm (Mr.SPA) to extract criminal supporters, which assigns a malicious relevance score (MRS) to each Twitter account to quantify how closely this account follows criminal accounts. In this section, we propose a Malicious Score Inference Algorithm (MSIA), which adapts and improves Mr.SPA [136] to assign a malicious score for each file to quantify its representativeness.

Given a constructed file relation graph $G = (V, E)$, let n be the number of nodes (files) in the graph, and $I(v_i, v_j)$ be the indicator to denote whether $(v_i, v_j) \in E$ (i.e., if $(v_i, v_j) \in E$, $I(v_i, v_j) = 1$; otherwise, $I(v_i, v_j) = 0$). At each step, for each node v_i, its malicious score $M(v_i)$ can be calculated as [136]

$$M(v_i) = \alpha \cdot \sum_{j=1}^{n} I(v_i, v_j) W(v_i, v_j) M(v_j), \tag{3.17}$$

where α is an adjustable factor, and $W(v_i, v_j)$ is the weight between v_i and v_j which reflects the coordination between each pair of nodes. For each node v_i, we calculate the similarity between itself and each of its neighbors v_j based on their presented features described above, denoted as $sim(Fv_i, Fv_j)$. Then, the weight $W(v_i, v_j)$ between node v_i and v_j is computed as

$$W(v_i, v_j) = \frac{sim(Fv_i, Fv_j)}{\sum_{e_{v_k v_j} \in E} sim(Fv_k, Fv_j)}. \tag{3.18}$$

In our application, we initialize $M^0(v_i) = \{0, 1\}$ (i.e., if v_i is malicious, $M^0(v_i) = 1$; otherwise, $M^0(v_i) = 0$). Through this malicious score propagation, (1) a file should sum up the weighted malicious scores inherited from the neighbors, and (2) the malicious score that a file receives from others should be dampened by the adjustable factor α [136].

With the consideration of the historical score record for each node, at each step $t(t \geq 1)$, an initial score bias $(1 - \alpha) \cdot M_i^0$ is added to its malicious score. Thus the malicious score vector $\overrightarrow{M^t}$ for all nodes at step $t(t \geq 1)$ can be computed as [136]

$$\overrightarrow{M^t} = \alpha \cdot \overrightarrow{\vec{I} \cdot \vec{M}^{t-1}} + (1 - \alpha) \cdot \overrightarrow{M^0}. \tag{3.19}$$

The algorithm MSIA stops when the updates of malicious score vector converge or a maximum number of the iterations has finished, and then we can obtain final malicious scores for all files. A threshold will be accordingly specified to determine the important and representative malware. The higher the malicious score it has, the more important the file is.

Belief propagation for malware detection. After we recognize the important and representative malware from the unknown file collection, based on *Finding 1* which states a file's goodness or malice not only depends on its neighbors, but also indirectly on other files (e.g., its neighbors' neighbors), we further apply our proposed EBP algorithm to detect the remaining malware, since EBP algorithm can propagate the indirect influences from other files for each node. The implementation of our proposed active learning framework is illustrated in Algorithm 2.

Algorithm 2: ***MSIA+EBP*** - An active learning framework for malware detection

---

Input: $G = (V, E)$: undirected weighted file relation graph(s), α: adjustable factor

Output: class label of each file

Initialize (file states, malicious score $\overrightarrow{M^0}$, $t = 1$): if v_i is malicious, $M^0(v_i) = 1$; otherwise, $M^0(v_i) = 0$;

Calculate the similarity $sim(Fv_i, Fv_j)$ for each pair of nodes v_i and v_j;

Calculate the weights Ws for each pair of nodes in G;

while *malicious scores haven't converged or iteration hasn't finished* **do**

  Calculate $\overrightarrow{M^t}$;

  $t = t + 1$

end

Label k files with the highest malicious scores (> threshold) as malware;

Use EBP (Algorithm 1) to label the remaining files;

3.3.3 Experimental Results and Analysis

In this section, we conduct three sets of experiments based on the collected dataset obtained from Comodo Cloud Security Center: (1) In the first set of experiments, we use MSIA to evaluate the effectiveness of the designed features; (2) In the second set

of experiments, we further evaluate our proposed active learning framework in malware detection; (3) In the last set of experiments, we compare our proposed framework with other classification methods (i.e., SVM, Decision Tree, and Naïve Bayes). We measure the malware detection performance of different methods using the evaluation measures shown in Table 3.5. All the experiments are conducted under the environment of 64 Bit Windows 7 operating system with 4th Generation Intel Core i7 Processor (Quad Core, 8MB Cache, up to 4.0GHz w/ Turbo Boost) plus 16G of RAM using Apache Pig, MySQL and C++.

Evaluation of the Designed Features

For each sample in the constructed file-to-file relations, we extract its relations with other samples described in Section 3.1 and its graph-based features designed in Section 3.3.1 for representation. In this section, we conduct the experiments using MSIA to evaluate the effectiveness of the designed features. The collected dataset from Comodo Cloud Security Center contains 60,724 files: 9,893 are malware, 19,402 are benign files, and 31,429 are unknown (with the analysis by anti-malware experts of Comodo Security Lab, 470 of them are labeled as malware and 1,273 of them are benign files). Those 9,893 malware and 19,402 benign files are used for training, while 470 malware and 1,273 benign files from the unknown file collection which are labeled by anti-malware experts are used for testing. The results in Table 3.7 demonstrate the effectiveness of the designed features in malware detection: though the graph-based features (GF) perform worse than the relations formulated by neighborhood (RF), the concatenation of these two types of features can significantly improve the detection performance.

Table 3.7: Evaluation of the designed graph-based features

Training	TP	FP	TN	FN	ACC
MSIA(RF)	7,392	1,301	18,101	2,501	0.8702
MSIA(GF)	6,960	2,410	16,992	2,933	0.8176
MSIA($RF+GF$)	7,890	1,371	18,031	2,003	0.8848
Testing	**TP**	**FP**	**TN**	**FN**	**ACC**
MSIA(RF)	293	78	1,195	177	0.8537
MSIA(GF)	179	122	1,151	291	0.7631
MSIA($RF+GF$)	315	78	1,195	155	0.8663

RF denotes the file relation features of the samples, while GF denotes the graph-based features of the samples.

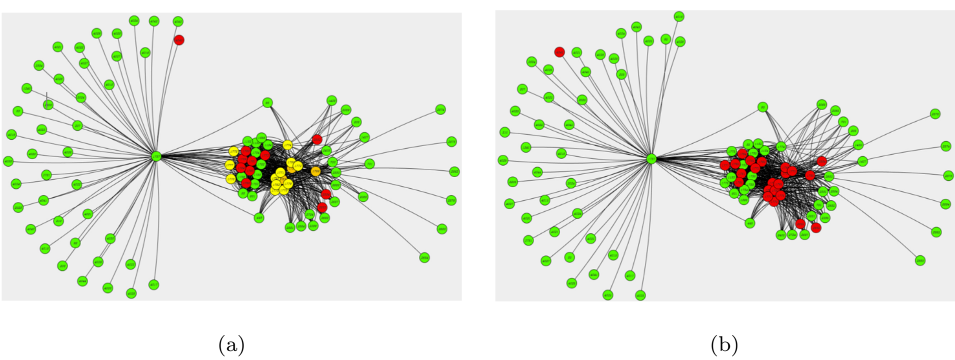
Evaluation of the Proposed Learning Framework

In this section, we further evaluate our proposed active learning framework in malware detection: (1) Based on the training and testing sets described in the previous section, we compare the performance of MSIA and EBP in malware detection; (2) To further improve the detection accuracy, we first apply MSIA for representative samples selection (193 samples are selected from the unknown file collection for labeling), and then use EBP for detection. The results in Table 3.8 show that our proposed framework composed of MSIA and EBP (MSIA+EBP) can greatly improve the accuracy in malware detection, compared with using MSIA and EBP respectively, or EBP after randomly selecting 193 samples from the unknown file collection for labeling (Random+EBP).

Table 3.8: Evaluation of the proposed learning framework in malware detection

Training	TP	FP	TN	FN	ACC
MSIA	7,890	1,371	18,031	2,003	0.8848
EBP	9,881	866	18,536	12	0.9700
Random+EBP	10,059	870	18,545	14	0.9701
MSIA+EBP	10,060	851	18,564	13	0.9707
Testing	**TP**	**FP**	**TN**	**FN**	**ACC**
MSIA	315	78	1,195	155	0.8663
EBP	411	119	1,154	59	0.8979
Random+EBP	462	204	1,069	8	0.8784
MSIA+EBP	437	100	1,173	33	0.9236

The example shown in Figure 3.13 further illustrates that, with three representative samples (orange ones in Figure 3.13(a)) selected, the related unknown files (yellow ones in Figure 3.13(a)) are correctly classified. Figure 3.13(b) shows the final detection results.



(a) (b)

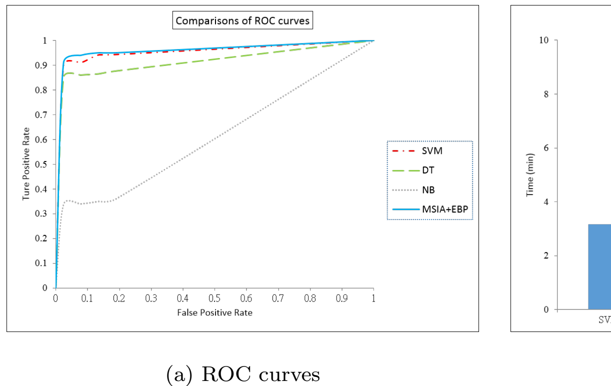
Figure 3.13: An example of malware detection using active learning framework.

Comparisons with Other Alternative Detection Methods

In this section, we further compare our proposed framework with other classification methods (Support Vector Machine (SVM), Decision Tree (DT), and Naïve Bayes (NB)) resting upon the same testing dataset described in the previous section. The results in Table 3.9 and the ROC curves for the cross-validation experiments based on the testing set in Figure 3.14(a) demonstrate that our proposed framework composed of MSIA and EBP (MSIA+EBP) is superior to SVM, DT, and NB in malware detection. Figure 3.14(b) shows that the detection efficiency of our proposed algorithms outperform other classification methods. The success of MSIA+EBP lies in the proper consideration and accommodation of the property of active learning, and the advantage of semi-supervised framework that makes use of labeled and unlabeled data for training.

Table 3.9: Comparisons of different detection methods

Training	TP	FP	TN	FN	ACC
SVM	8,661	797	18,618	1,412	0.9251
DT	8,308	1,761	17,654	1,765	0.8804
NB	5,288	502	18,913	4,785	0.8207
MSIA+EBP	10,060	851	18,564	13	0.9707
Testing	TP	FP	TN	FN	ACC
SVM	452	172	1,101	18	0.8910
DT	412	241	1,032	58	0.8285
NB	159	31	1,242	311	0.8037
MSIA+EBP	437	100	1,173	33	0.9236



(a) ROC curves

(b) Detection Efficiency

Figure 3.14: Comparisons of ROC curves and detection efficiency of different methods

3.4 Graph Representation Learning for Malware Detection

Despite the BP algorithm can propagate the indirect influence from other files, it merely preserves the graph structure information by considering short and fixed neighborhood information, i.e., the first and second order proximities, which cannot capture long-range structure over file-to-file relation graph. To address this issue, in this section, we present a sequence modeling method *file2vec* to learn representations for files over graph to capture more meaningful proximity: given a set of file sequences generated using random walk, a seq2seq model [120] Long Short-term Memory (LSTM) [120, 6, 35] is introduced to read the input file sequence to obtain a fixed-length summary vector from which another LSTM is employed to generate the output sequence (i.e., encoding and decoding the file sequences), through which the fixed-dimensional representation for each file will be learned.

3.4.1 Representation Learning using Long Short-term Memory

Given a graph $G = (V, E)$, the ***graph representation learning*** task is to learn a function $f : V \rightarrow \mathbb{R}^d$ that maps each node $v \in V$ to a vector in a d-dimensional space $\mathbb{R}^d$, $d \ll |V|$ that is capable to preserve the structural relations among them. In our application, the files in the constructed file relation graph can be connected through different number of nodes (i.e., files) and edges (i.e., co-occurrence relations). For example, as shown in Figure 3.15, file B4 can be connected to file B5 through *Seq2*, and connected to file M2 through *Seq4* as well. It's recalled that Belief Propagation updates a node's message from its neighbor node considering all the messages flowing into its neighbor node; this method only takes consideration of the first and second order neighbors while fails to directly learn the long-range relatedness between files like B4 and M2, and B5 and M2. To fully capture the graph structure information to represent each file over graph, this calls for a new method for representation learning.

Since the graph structure information is preserved by file sequences (e.g., *Seq1*-*Seq4* in Figure 3.15), the sequence modeling method can seamlessly fuse the long-range graph structure information into the final learned representations [86]. The sequence to sequence models (seq2seq) have been successfully applied to machine translation [6, 35], and Natural Language Processing (NLP) problems [55] in recent years. The rationale using seq2seq for representation learning is to deploy a Long Short-term Memory (LSTM) to read the input sequence, one at each timestep, to obtain an overall sequence vector representation, and then deploy another LSTM to extract the output sequence from

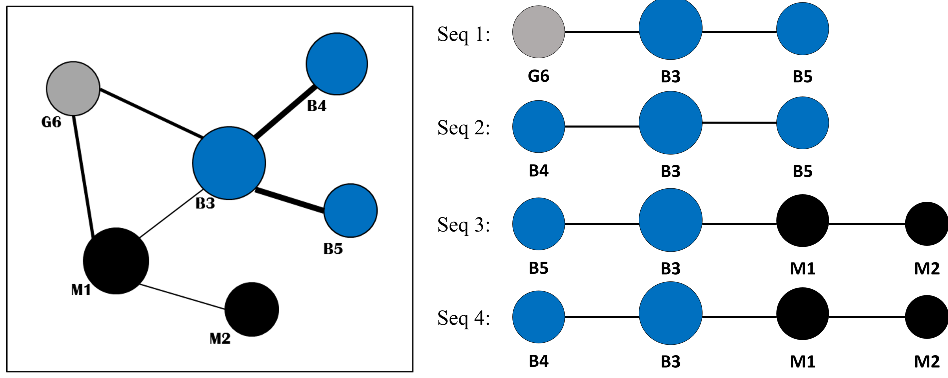


Figure 3.15: Neighborhood relationships among files.

that vector [86]. The structure information is seamlessly incorporated into the latent vectors of hidden layers, which can be effectively used as the representations of nodes. In the following, we will introduce how to use *file2vec* for representation learning of files over graph. We will first present file sequence generation using random walk, and then leverage LSTM for the generated file sequence modeling.

File sequence generation using random walk. Given a source node v_j in a graph, the random walk is a stochastic process with random variables $v_j^1, v_j^2, ..., v_j^k$ such that v_j^{k+1} is a node chosen at random from the neighbors of node v_k. The transition probability $p(v_j^{i+1}|v_j^i)$ at step i is the normalized probability distributed over the neighbors of v_j^i, which can be denoted as:

$$p(v_j^{i+1}|v_j^i) = \frac{1}{|N(v_j^i)|}, \tag{3.20}$$

where $N(v_j^i)$ denotes the neighborhood of node v_j^i. The walk paths (i.e., sequences) generated by the above strategy are able to preserve structural relations between different nodes in the graph, and thus will facilitate the representation learning using LSTM.

File sequence modeling using LSTM. A LSTM is an architecture designed for recurrent neural network to address the vanishing/exploding gradient issue [120, 64]. In general, LSTM learns a mapping from an input sequence (i.e., $(\mathbf{x}_1, ..., \mathbf{x}_T)$, where $\mathbf{x}_t \in \mathbb{R}^n$ is a vector at timestep t) to an output sequence. As intermediate output, LSTM generates a vector $\mathbf{h}_t \in \mathbb{R}^d$ for each timestep. By furthering pooling all the $\mathbf{h}_t$'s, we can output the embedding vectors. In our application, given an input file sequence $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_k)$ where $\mathbf{v}_t \in \mathbb{R}^{|V|}$ is a $|V|$-dimensional one-hop vector at timestep t, we will employ an encoder-decoder LSTM architecture [35] (as illustrated in Figure 3.16)

for file sequence modeling, in which hidden layer vectors are elaborately extracted as the representations for the corresponding files to improve the quality of representation learning.

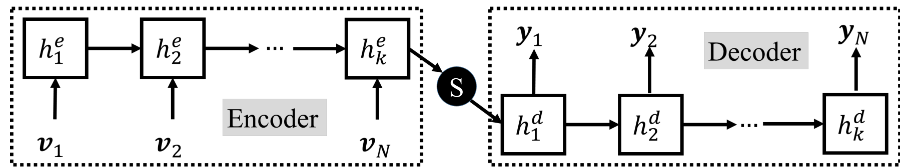


Figure 3.16: Illustration of encoder-decoder LSTM architecture.

Encoder: The LSTM encodes the input file sequence $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_k)$ through the hidden layer function $\mathcal{H}$ so that each hidden layer vector $\mathbf{h}_t^e$ at timestep t can be denoted as

$$\mathbf{h}_t^e = \mathcal{H}(\mathbf{v}_t, \mathbf{h}_{t-1}^e), \tag{3.21}$$

where $\mathcal{H}$ is implemented using purpose-built memory cells to store information, which can be formulated as the following composite functions [55]:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{v}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1}^e + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \tag{3.22}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{v}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1}^e + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f) \tag{3.23}$$

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tanh(\mathbf{W}_{xc}\mathbf{v}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1}^e + \mathbf{b}_c) \tag{3.24}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{v}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1}^e + \mathbf{W}_{co}\mathbf{c}_{t-1} + \mathbf{b}_o) \tag{3.25}$$

$$\mathbf{h}_t^e = \mathbf{o}_t \circ \tanh(\mathbf{c}_t) \tag{3.26}$$

where σ is the logistic sigmoid function, $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$, $\mathbf{c}_t$ are the input gate, forget gate, output gate, and cell activation vectors respectively, $\mathbf{W}$s are the weight matrices, $\mathbf{b}$s are the bias vectors, and $\circ$ is the point-wise product between two vectors. After reading $\mathbf{v}_k$, the hidden state $\mathbf{h}_k^e$ is used as the summary vector $\mathbf{s}$ of the whole input sequence.

Decoder: The summary vector $\mathbf{s}$ is fed back into the LSTM's first hidden layer so that $\mathbf{h}_0^d = \mathbf{s}$, and then each hidden layer vector $\mathbf{h}_t^d$ at timestep t can be calculated as

$$\mathbf{h}_t^d = \mathcal{H}(\mathbf{0}, \mathbf{h}_{t-1}^d), \tag{3.27}$$

where $\mathbf{0}$ is an all-zero vector. Accordingly, the output vector $\mathbf{y}_t \in \mathbb{R}^{|V|}$ can be formulated as follows [55]:

$$\mathbf{y}_t = \sigma(\mathbf{W}_{hy}\mathbf{h}_t^d + \mathbf{b}_y). \tag{3.28}$$

$\mathbf{y}_t$ is capable to predict the real file v_t through a softmax layer. The sequence loss $\mathcal{L}$ is adopted to measure the correctness of decoding, which is computed as

$$\begin{aligned} \mathcal{L} &= -\sum_{t=1}^{k} \log p(\mathbf{v}_t|\mathbf{y}_t) \\ &= -\sum_{t=1}^{k} \log \frac{\exp(y_t^{v_t})}{\sum_{i=1}^{|V|} \exp(y_t^{v_i})}. \end{aligned} \tag{3.29}$$

The weights can be efficiently calculated with backpropagation through time [130, 55], and the LSTM model can then be trained using Adam optimization algorithm.

For the generated file sequences, each file may appear in multiple sequences. Suppose that file v_i exists in $|v_i|$ sequences, by doing concatenation of max and avg pooling over all $\mathbf{h}_j^e$'s for file v_i, $\forall j = 1, 2, ..., |v_i|$, we obtain an embedding $\mathbf{h}$ for each file:

$$\mathbf{h} = maxavgPooling(\{\mathbf{h}_j^e : j = 1, ..., |v_i|\}). \tag{3.30}$$

Using *file2vec*, the mapped feature vectors of files, encoding the informaiton of graph structure, can be fed to a classifier to train the classification model, based on which the unlabeled files can be predicted if they are malicious or not. The implementation of *file2vec* is illustrated in Algorithm 3.

Algorithm 3: *file2vec* - A graph representation learning model for malware detection

---

Input: $G = (V, E)$, walks per node r, length l, and vector dimension d, training data set D_t, testing data set D_e

Output: class label of each file

for $i = 1 \rightarrow |V|$ **do**

  for $j = 1 \rightarrow r$ **do**

    get l-length random walk path using Eq. 3.20;

  end

  Use LSTM to model paths generated and output $\mathbf{h}^e \in \mathbb{R}^d$;

end

$\mathbf{h} = maxavgPooling(\{\mathbf{h}_j^e : j = 1, ..., |v_i|\})$ for each file;

Train SVM using $\mathbf{h}_{D_t}$;

for $n = 1 \rightarrow |D_e|$ **do**

  Generate the label using trained SVM;

end

3.4.2 Experimental Results and Analysis

In this section, we conduct three sets of experimental studies using the same data collected from Comodo Cloud Security used in Section 3.3.3 to fully evaluate the performance of our proposed representation learning method *file2vec* in malware detection. We use the same performance indices shown in Table 3.5.

Evaluation of *file2vec*

In this set of experiments, we evaluate our proposed method *file2vec* by comparisons with another popular representation learning method DeepWalk [104] using random walk and skip-gram. The parameter settings used for *file2vec* are in line with typical values used for the baseline: vector dimension $d = 200$, walks per node $r = 15$, and walk length $l = 50$. To facilitate the comparisons, we randomly select a portion of labeled files (ranging in {10%, 30%, 50%, 70%, 90%}) from 9,893 malware and 19,402 benign files for training and the remaining ones for testing to evaluate their performances. The SVM is used as the classification model for both DeepWalk and *file2vec*.

Table 3.10 illustrates the detection results of different representation learning methods. From Table 3.10, we can see that DeepWalk performs slightly better than *file2vec* when the portion of training data is 10% and 50%, but the difference is not statistically significant; when the training data reaches to 70%, and 90%, *file2vec* performs better for malware detection in terms of *ACC* and *F1*. That is to say, *file2vec* learns significantly better file representation than current state-of-the-art method. The success of *file2vec* stems from the sophisticated sequence modeling, which leverages the advantage of long-range graph structure. More importantly, DeepWalk assigns each file a static embedding vector based on all sequences, while LSTM reads the whole input sequence to further generate the output sequence, and its learned representations tend to be context-aware to different sequences it interacts with.

Table 3.10: Comparisons of *file2vec* with DeepWalk in malware detection

Metric	Method	10%	30%	50%	70%	90%
ACC	DeepWalk	0.7482	0.8029	0.8541	0.9121	0.9267
	file2vec	0.7380	0.8101	0.8424	0.9221	0.9438
F1	DeepWalk	0.5701	0.6710	0.7606	0.8646	0.8890
	file2vec	0.5500	0.6864	0.7395	0.8829	0.9167

Comparisons with Other Alternative Methods

In this set of experiments, based on the dataset used in Section 3.3.3, we compare *file2vec* with other alternative machine learning methods. For these methods, we construct three types of features: *f–1*: relation-based features (i.e., file co-occurrence used in Section 3.2); *f–2*: concatenation of relation-based features and graph-based features (i.e., F_v introduced in Section 3.1); *f–3*: representations learned by *file2vec*. Based on these features, we consider five classification models, i.e., NB on *f–1* and *f–2*, SVM on *f–1* and *f–2*, EBP on *f–1*, MSIA+EBP on *f–2*, and *file2vec* on *f–3*. The experimental results are illustrated in Table 3.11. From the results we can observe that feature engineering (*f-2*: concatenation of relation- and graph- based features) helps the performance of machine learning, and active learning also facilitates malware detection, but *file2vec* that encodes the graph structure and the long-range influence among files learned from LSTM significantly outperforms other baselines. This again demonstrates that, to detect malware, *file2vec* using sequence modeling is able to achieve better detection performance.

Table 3.11: Comparisons with other machine learning methods

Metric	NB		SVM		EBP	MSIA+EBP	*file2vec*
	f-1	*f-2*	*f-1*	*f-2*			
ACC	0.8037	0.8101	0.8909	0.8967	0.8978	0.9236	0.9454
F1	0.4818	0.5052	0.8263	0.8348	0.8220	0.8679	0.9023

Evaluation of Parameter Sensitivity

In this set of experiments, based on the dataset used in Section 3.3.3, we conduct the *sensitivity* analysis of how different choices of parameters (i.e., walks per node r, walk length l, and vector dimension d) will affect the performance of *file2vec* in malware detection. From the results shown in Figure 3.17(a) and 3.17(b), we can observe that the balance between computational cost (number of walks per node r and walk length l in x-axis) and efficacy (F1 in y-axis) can be achieved when $r = 15$ and $l = 60$ for malware detection. We also examine how vector dimensions (d) affect the performance. As shown in Figure 3.17(c) we can find that the performance inclines to be stable when d increases to around 300. Overall, *file2vec* is not strictly sensitive to these parameters, and is able to reach high performance under a cost-effective parameter choice.

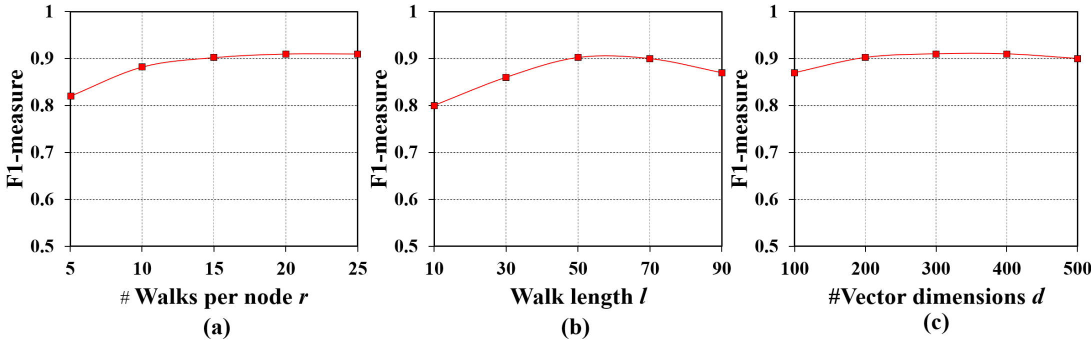


Figure 3.17: Parameter sensitivity evaluation.

3.5 Summary

In this chapter, we provide deep analysis of file-to-file relations between malware and benign files and study how the file co-existence relation graphs can be constructed. Resting on the constructed file-to-file relation graphs, we first design an enhanced Belief Propagation algorithm for unknown file labeling that fine tunes various components used in the algorithm and formulates the new message and belief read-out functions; then we investigate several new and robust graph-based features for malware detection and reveal the characteristics of file relations, based on which we propose an effective active learning framework (MSIA+EBP) for malware detection; last, we leverage a sequence modeling method Long Short-term Memory to learn the representations of files in our constructed graph which captures the long-range structural information. To the best of our knowledge, this is the first investigation of the relationship characteristics for the file-to-file relations in malware detection using social network analysis. Due to the difficulty in thoroughly obtaining the social interactions and motivations of malware, we recognize that the validations on some proposed explanations are not entirely rigorous. However, we believe that our novel analysis of those phenomena still yields great value and unveils a new avenue for better understanding malware's file relation ecosystem. The research work conducted in this chapter have been also published in the following papers:

- Lingwei Chen, William Hardy, Yanfang Ye ✉, Tao Li. "Analyzing File-to-File Relation Network in Malware Detection", International Conference on Web Information Systems Engineering (WISE), 415–430, 2015.

- Lingwei Chen, Tao Li, Melih Abdulhayoglu, Yanfang Ye ✉. "Intelligent Malware Detection Based on File Relation Graphs", IEEE International Conference on Se-

matic Computing (ICSC), 85–92, 2015.

- Shifu Hou, Lingwei Chen, Yanfang Ye ✉, Lifei Chen. "Deep Analysis and Utilization of Malware's Social Relation Network for Its Detection", Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data, 31–42, 2017.

- Yanfang Ye ✉, Shifu Hou, Lingwei Chen, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, Fudong Shao. "AiDroid: When Heterogeneous Information Network Marries Deep Neural Network for Real-time Android Malware Detection", arXiv preprint arXiv:1811.01027, 2018.

Chapter 4

Enhancing Security of Learning-based Systems in Malware Detection

The existing works [149, 69, 47] and the work in the previous section have demonstrated that relation-based features integrated with file contents are more resilient against malware attacks compared to content-based only representations. However, as machine learning-based detection systems become more widely deployed, the adversary incentive for defeating them increases. Therefore, we go further insight into the arms race between adversarial malware attack and defense, and aim to enhance the security of machine learning-based detection systems. In this chapter, we focus on the studies on the following research questions:

- *How can we define adversarial malware attack?*
- *In response to the adversary's strategy, how can we design an adversary-aware learning model based on the skills and capacities of the attackers to enhance the security of machine Learning-based malware detection?*
- *Since it is computationally expensive and almost impossible to find all adversarial models, can we design defensive learning models whose action space is practically independent from the attacks?*

On the basis of a learning-based classifier with the input of different feature representations extracted from the Portable Executable (PE) files and Android application (app) files respectively, we investigate the adversarial malware attacks and aim to enhance

security of machine learning-based detection against such attacks. In particular, we first explore the adversarial attacks corresponding to the different scenarios, thoroughly assess the adversary behaviors through feature manipulations, adversarial cost, and attack goals, and accordingly present a general attack strategy. Resting on the learning-based classifier which is degraded by the adversarial malware attacks, we propose three secure-learning paradigms *SecDefender*, *SecureDroid*, and *Droideye*, either depending on or independent from the skills and capabilities of the attackers, to counter these adversarial attacks, and thus enhance the security of the classifier while not compromising its detection accuracy. The proposed methods can be readily applied in other malware detection tasks.

4.1 Problem Definition

Machine Learning-based Classifier for Malware Detection. A malware detection system using machine learning techniques attempts to identify variants of known malware or zero-day malware through building a classification model based on the labeled training samples and predefined feature representations, which is illustrated in Figure 4.1. More specifically, the problem of machine learning-based malware detection

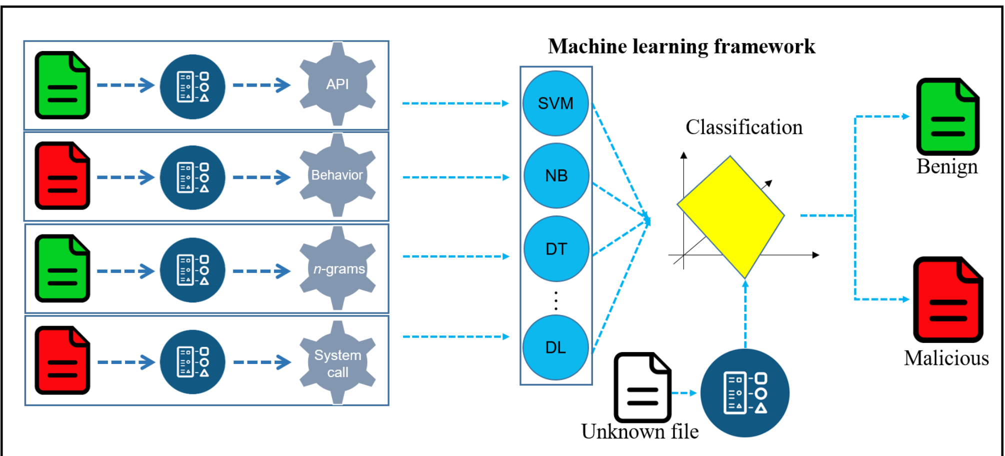


Figure 4.1: Intelligent malware detection system using machine learning techniques.

can be stated in the form of: $f : \mathcal{X} \to \mathcal{Y}$ which assigns a label $y \in \mathcal{Y}$ (i.e., -1 or $+1$) to an input file sample $\mathbf{x} \in \mathcal{X}$ through the learning function f. A general linear classification model for malware detection can be thereby denoted as:

$$\mathbf{f} = \text{sign}(f(\mathbf{X})) = \text{sign}(\mathbf{X}^T\mathbf{w} + \mathbf{b}), \tag{4.1}$$

where $\mathbf{f}$ is a vector, each of whose elements is the label (i.e., malicious or benign) of a file to be predicted, each column of matrix $\mathbf{X}$ is the feature vector of a file, $\mathbf{w}$ is the weight vector and $\mathbf{b}$ is the biases. Typically, a machine learning system on the basis of a linear classifier can be formalized as an optimization problem [149]:

$$\underset{\mathbf{f},\mathbf{w},\mathbf{b};\boldsymbol{\xi}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{y}-\mathbf{f}||^2 + \frac{1}{2\beta}\mathbf{w}^T\mathbf{w} + \frac{1}{2\gamma}\mathbf{b}^T\mathbf{b} + \boldsymbol{\xi}^T(\mathbf{f}-\mathbf{X}^T\mathbf{w}-\mathbf{b}), \tag{4.2}$$

subject to Eq. (4.1), where $\mathbf{y}$ is the labeled information vector, $\boldsymbol{\xi}$ is Lagrange multiplier which is a strategy for finding the local minima of $\frac{1}{2}||\mathbf{y}-\mathbf{f}||^2$ subject to $\mathbf{f}-\mathbf{X}^T\mathbf{w}-\mathbf{b}=0$, β and γ are the regularization parameters, and $\frac{1}{2\beta}\mathbf{w}^T\mathbf{w}$ and $\frac{1}{2\gamma}\mathbf{h}^T\mathbf{h}$ are regularization terms to deal with the overfitting problem in the learning model. Note that Eq. (4.2) is a general linear classifier (denoted as *Original-Classifier* throughout the chapter) consisting of specific loss function and regularization terms. Without loss of generality, the equation can be transformed into different linear models depending on the choices of loss function and regularization terms, such as Logistic Regression (LR) and Support Vector Machine (SVM).

Security Violation. In malware detection, the learner's purpose is to classify malware and prevent them from interfering users' computers. In contrast, adversaries would like to violate the security context by either (a) allowing malicious files to be misclassified as false negatives (an integrity attack) or (b) creating a denial of service in which benign files are incorrectly classified as false positives (an availability attack). In other words, there are two types of security violations the adversaries cause [10, 11]: (1) *Evasion attack* (also called integrity attack) manipulates malicious samples at test time to have them misclassified as benign without having influence over the training data; (2) *Poisoning attack* (also called availability attack) injects poisoning samples into the training data to create a denial of service that disables benign files being normally executed. In this dissertation, we focus on the former attack. We call evasion attack as adversarial attack throughout the dissertation. Security violation can be targeted or indiscriminate, depending on whether the attacker is interested in having some specific malware misclassified, or if any misclassified malware sample meets his/her goal [41].

4.2 Adversarial Attack

Adversarial attacks can generally be modeled as an optimization problem: given an original malicious file $\mathbf{x} \in \mathcal{X}^+$, the adversarial attacks attempt to manipulate its

features to be detected as benign (i.e., $\mathbf{x}' \in \mathcal{X}^-$), with the minimal adversarial cost. In this section, we present how attackers can achieve such attacks.

Considering that the attacker may have different levels of knowledge of the targeted learning system [126], he may know completely, partially, or do not have any information about: (i) the feature extraction method, (ii) the training sample set, and (iii) the learning algorithm. We characterize the attacker's knowledge in terms of a space Ψ that encodes knowledge of the feature space X, the training sample set D, and the classification function f. In the traditional detection evasions, we mainly discuss the scenario that the attackers using techniques such as encryption, obfuscation, and polymorphism to probe the classifier without any knowledge of the learning system (i.e., $\Psi = ()$). In this section, we wish to follow the common practice in cyber security research of erring on the side of overestimating the attackers' capabilities rather than underestimating them. Therefore, based on the different scenarios, we present three well-defined adversarial attacks to facilitate security analysis of the classifier as below.

Mimicry Attacks In this scenario, the attackers are assumed to know the feature space and be able to obtain a collection of malware samples and benign files to imitate the original training dataset. In other words, $\Psi = (X, \hat{D})$. In such attack, the strategy of the attackers is to manipulate a set of features (e.g., Windows API calls) to probe the learning system. The effectiveness of this adversarial attack mainly depends on the similarity of distribution between the original training dataset and mimic dataset. It's more likely that the attackers may evade the detection if the file samples drawn from surrogate dataset are distributed closely as the training sample set.

Imperfect-knowledge Attacks Further than the previous scenario, we assume that both the feature space and the original training sample set can be fully controlled by the attackers, i.e., $\Psi = (X, D)$. Compared with mimicry attacks, the knowledge of the malware and benign files in the original training dataset definitely leverage clearer insight for the attackers to conduct the adversarial attacks to evade the learning system's detection, although they may have no knowledge of the learning algorithm.

Ideal-knowledge Attacks This is the worst case where the learning algorithm is also known to the attackers, i.e., $\Psi = (X, D, f)$. Although many settings do impose significant restrictions on getting the ideal knowledge by the attackers, including the feature space, the training sample set and the classification function, we would like

to overestimate the attackers' capabilities rather than underestimate them. When the attackers can perfectly access to the learning system, they can thoroughly analyze the malware and benign files in the training dataset, investigate the vulnerability of the classification algorithm, and accurately manipulate the features to evade the detection. All these advantages contribute to an effective adversarial attack, which has the strongest probability of evading the targeted learning system. Since this worst case provides a potential upper bound on the performance degradation suffered by the learning system under the adversarial attacks, it can be used as reference to evaluate the effectiveness of the learning system under the other simulated attack scenarios.

Figure 4.2 depicts the aforementioned adversarial attacks according to different levels of knowledge the attackers may have.

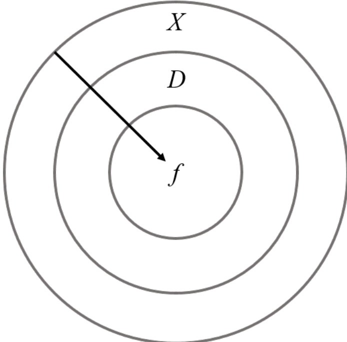


Figure 4.2: Different scenarios of the adversarial attacks. With the direction of the inward arrow, the adversarial attacks are depicted with the knowledge of $(X, \hat{D})$, (X, D), and (X, D, f).

4.2.1 Feature Manipulation

To conduct an adversarial attack, attackers would manipulate the features of a malicious file to evade the detection. Feature manipulation defines how malware samples can be modified, according to program-specific constraints [41]; these constraints can be encoded in terms of distances in feature space, computed between the source malware data and its manipulated versions, which will be discussed in the next section. Given a file, after feature extraction, it can be represented by a binary feature vector. Then a typical manipulation can be either adding or eliminating a binary in the vector.

- **Feature Addition.** In this scenario, attackers can autonomously inject a feature in the file (i.e., set 0 to 1). For example, they can add API calls in a file without

influence on other existing functionalities; they can also inject API calls in a dead code or methods which will be never called by any invoke instructions in a file.

- **Feature Elimination.** In this setting, attackers may hide or remove a feature from the file (i.e., set 1 to 0) while not affecting the intrusive functionality they want to execute. For example, attackers can hide the information stored as strings by encryption and decrypting it at runtime.

Either feature addition or elimination, both settings should retain the semantics and intrusive functionality of the original file after manipulations. In such case, feature addition is easier and safer when the injection is not directly executed by the file (as examples shown above). However, if attackers want to inject a suspicious API call to the file being executed by the program, it will be more sophisticated and may influence the semantics of the file. Feature elimination is usually more complicated, such as, removing API calls from a file is not always practical since it may limit the functionalities of the file. Therefore, conducting an adversarial attack that needs to manipulate a lot of features while not compromising the malicious functionalities may not always be feasible. In this respect, attackers may need to implement a well-crafted attack by taking consideration of the adversarial cost.

4.2.2 Adversarial Cost

The adversarial cost can be defined in terms of distances in feature space between the original malware and its manipulated version; simply, it can be decided by the number of binaries that are changed from $\mathbf{x}$ to $\mathbf{x}'$ by attackers, which is denoted as

$$\mathcal{C}(\mathbf{x}', \mathbf{x}) = ||\mathbf{c}^T(\mathbf{x}' - \mathbf{x})||_p^p, \tag{4.3}$$

where $\mathbf{c}$ is a vector whose element denotes the corresponding cost of changing a feature, and p is a real number. The adversarial cost function can be considered as ℓ_1-norm or ℓ_2-norm depending on the feature space. For attackers, the manipulation cost c_i for each feature is different. For example, some specific Windows API calls may affect the structure for intrusive functionality, which are more expensive to be modified. Therefore, the manipulation cost c_i for each feature is practically significant, which is determined by the feature type and manipulation method. Furthermore, for the reasons aforementioned, it's impractical for attackers to modify a malware into benign at any cost (i.e., manipulating a large number of features). For instance, an adware will automatically display

or download advertisements when the victim is online. The attacker will not modify its related API calls to make the adware being benign and loss its malicious functionalities. Thus, there is an upper limit of the maximum manipulations that can be made to the original malware $\mathbf{x}$. That is, the manipulation function $\mathcal{A}(\mathbf{x})$ can be formulated as

$$\mathcal{A}(\mathbf{x}) = \begin{cases} \mathbf{x}' & \text{sign}(f(\mathbf{x}')) = -1 \textit{ and } \mathcal{C}(\mathbf{x}', \mathbf{x}) \leq \delta_{\max} \\ \mathbf{x} & \text{otherwise} \end{cases}, \tag{4.4}$$

where the malware is manipulated to be misclassified as benign only if the adversarial cost is less than or equal to a maximum cost $\delta_{\max}$.

4.2.3 Attack Strategy

In practice, though attackers may know differently about the targeted learning system [126], they always have the following two competing objectives: (1) maximize the number of malicious files being classified as benign, and (2) minimize the adversarial cost for optimal attacks over the learning-based classifier [83]. Specifically, the adversarial attack strategy can be formulated as:

$$\underset{\mathbf{x}' \in \mathbf{X}^-}{\text{argmin}} \ \min\{f(\mathbf{x}'), 0\} + \mathcal{C}(\mathbf{x}', \mathbf{x}), \tag{4.5}$$

subject to $\mathcal{C}(\mathbf{x}', \mathbf{x}) \leq \delta_{\max}$.

Given an original malware, an effective adversarial attack generally modifies a small portion of features with the low adversarial cost. Let Eq. (4.5) return an optimal solution $\mathbf{x}^*$ with $\text{sign}(f(\mathbf{x}^*)) = -1$, and a suboptimal solution $\tilde{\mathbf{x}}$ with $\text{sign}(f(\tilde{\mathbf{x}})) = -1$, we characterize the relationship between $\mathbf{x}^*$ and $\tilde{\mathbf{x}}$, and have

$$\min\{f(\mathbf{x}^*), 0\} = \min\{f(\tilde{\mathbf{x}}), 0\} = -1.$$

The difference between $\mathbf{x}^*$ and $\tilde{\mathbf{x}}$ can be simplified as the comparison between their adversarial costs, i.e., argmin $\mathcal{C}(\mathbf{x}', \mathbf{x})$. According to the definition of the adversarial cost in Eq. (4.3), $\mathcal{C}(\mathbf{x}', \mathbf{x}) \geq 0$. $\mathcal{C}(\mathbf{x}', \mathbf{x}) = 0$ iff $\mathbf{x}' = \mathbf{x}$. $\mathcal{C}(\cdot)$ is then strictly convex in $\mathbf{x}'$ and has a unique solution for this optimization problem. Therefore, $\mathcal{C}(\mathbf{x}^*, \mathbf{x}) < \mathcal{C}(\tilde{\mathbf{x}}, \mathbf{x})$, since $\mathbf{x}^*$ is an optimal attack. The adversarial cost varies resting on the different levels of knowledge the attackers have about the targeted learning system. We formalize this relationship between $\mathbf{x}^*$ and $\tilde{\mathbf{x}}$ in the following lemma.

Lemma 4.1 *Suppose* $\mathbf{x}^*$ *is the optimal adversarial attack to Eq. (4.5), while* $\tilde{\mathbf{x}}$ *is suboptimal attack, s.t.,* $f(\mathbf{x}^*) < 0$ *and* $f(\tilde{\mathbf{x}}) < 0$. *Then* $\mathcal{C}(\mathbf{x}^*, \mathbf{x}) < \mathcal{C}(\tilde{\mathbf{x}}, \mathbf{x})$. *That is, the*

optimal adversarial attack can access to the learning-based system with minimum adversarial cost.

According to this lemma, to perform an optimal adversarial attack, attacker may want to select the features that are easy to be manipulated (e.g., addition is generally easier than elimination) and to choose the features that have higher contributions to the classification problem. This inspires us to design secure defenses to combat the adversarial attack strategy.

4.3 *SecDefender*: A Secure-learning Model against Well-crafted Attack

In this section, on the basis of Windows API calls extracted from the PE files, we first present a well-crafted adversarial attack model (named *AdvAttack*) to thoroughly assess the security of the classifier by considering different contributions of the API calls to the classification problem. To effectively counter such adversarial attacks, we further propose a resilient yet elegant secure-learning model (named *SecDefender*) based on *AdvAttack* for malware detection. The system architecture of *SecDefender* is shown in Figure 4.3.

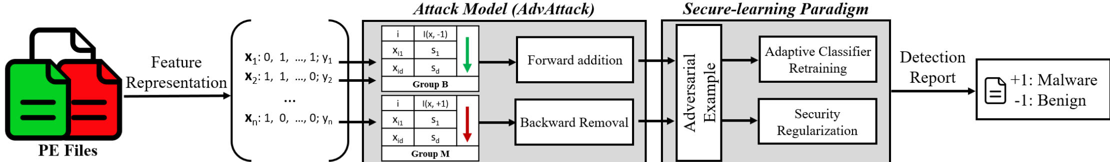


Figure 4.3: An overview of system architecture of *SecDefender*. In this system, the collected PE files are first represented as d-dimensional binary feature vectors. Then a well-crafted adversarial attack model *AdvAttack* is formulated to generate the adversarial examples, which will be further used for classifier retraining and security regularization. For a new file, based on the extracted features, it will be predicted as either malicious or benign based on the trained classification model.

4.3.1 Feature Representation

PE is designed as a common file format for all flavors of Windows operating system, and malicious PE files are in the majority of the malware in recent years [140]. Based on the collected PE file sample set, without loss of generality, in this section, we extract

Windows API calls as the features to represent the file samples, since they can effectively reflect the behaviors of program codes [140]. For example, the API "*GetFileType*" in "*KERNEL32.DLL*" can be used to retrieve the file type of the specified file, while the API "*GetDlgItemText*" in "*USER32.DLL*" is utilized to obtain the title or text associated with a control in a dialog box. Before feature extraction, if a PE file is previously compressed by a third party binary compress tool such as UPX and ASPack Shell or embedded a homemade packer, it will be decompressed at first and we use the dissembler CMDsm developed by Comodo Anti-malware Lab to dissemble the PE code and output the assembly instructions as the input for the Windows API call extraction.

In this section, we perform static analysis on the collected file samples and extract the above features (i.e., Windows API calls) to represent the files. Though static analysis has unequivocal limitations, since it is not feasible to analyze malicious code that is thoroughly obfuscated or decrypted at runtime. For this reason, considering such attacks would be irrelevant for the scope of our work. Our focus is rather to understand and to enhance the security properties of learning-based system against a wide class of adversarial attacks. The above features are exploited as a case study which facilitate the understanding of our further proposed approach, while other feature extractions, such as binary n-gram, dynamic system calls, and dynamic behaviors, are also applicable in our further investigation.

To represent each collected PE file, we first extract the features and convert them into a vector space, so that it can be fed to the classifier either for training or testing. Based on the extracted features, we denote our dataset D to be of the form $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$ of n file samples, where $\mathbf{x}_i$ is the set of features extracted from file i, and y_i is the class label of file i, where $y_i \in \{+1, -1, 0\}$ ($+1$ denotes malicious, -1 denotes benign, and 0 denotes unknown). Let d be the number of all extracted features in the dataset D. Each of the PE file can be represented by a binary feature vector:

$$\mathbf{x}_i = \begin{pmatrix} 0 \\ 1 \\ \dots \\ 1 \\ 0 \end{pmatrix} \rightarrow \left.\begin{matrix} \text{KERNEL32.DLL,VirtualQuery;} \\ \text{KERNEL32.DLL,Sleep;} \\ \dots \\ \text{USER32.DLL,DestroyIcon;} \\ \text{KERNEL32.DLL,SetEvent} \end{matrix}\right\} \text{Windows API calls} \tag{4.6}$$

where $\mathbf{x}_i \in \mathbb{R}^d$, and $x_{ij} = \{0, 1\}$ (i.e., if file i includes feature j, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$).

4.3.2 Well-crafted Attack Model *AdvAttack*

Characteristics of the Feature Set

Since it's the most important for the attackers to choose a relevant subset of features applied for addition and elimination, to well implement the attack, we take deep insight into the property of the feature set. As different features (i.e., API calls in our application) differently contribute to the classification of malware and benign files, it's worth to investigate the importance of each feature. We analyze the sample set obtained from Comodo Cloud Security Center, which contains 10,000 labeled files with $3,503$ extracted API calls. There are various methods for assessing feature relevance (e.g., information gain, χ^2 contingency table statistic, etc.) in classification, each of which has its own pros and cons [59]. Here we use Max-Relevance algorithm [103], which is one of the popular approaches to define dependency of variables and has also been successfully applied in malware detection [147], to calculate the relevance score of each API call for the classification of malware and benign file respectively. Given x representing an API call, and the file label y, their mutual information is defined in terms of their frequencies of appearances $P(x)$, $P(y)$, and $p(x, y)$ as follows [147]:

$$I(x, y) = \int \int p(x, y) \log \frac{p(x, y)}{P(x)P(y)} dxdy. \tag{4.7}$$

Figure 4.4 shows the distribution of the relevance scores of the extracted API calls for the classification of malware and benign files, from which we can see that for those with high relevance scores, some are explicitly relevant to malware, while some have high influence on the classification of benign files. Note that API calls with extremely low relevance scores (about 85% lower than 0.0005) have limited or no contributions in malware detection (e.g., *SetLocalTime* in *KERNEL32.DLL*), thus they will not be considered for the further investigated adversarial attacks.

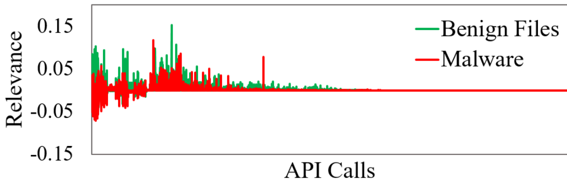


Figure 4.4: Relevance score distribution of the extracted API calls for the classification of malware and benign files

To further analyze the different importances of API calls for the classification of

malware and benign files, we take insights into their specific functionalities. Table 4.1 shows the top ranked API calls related to malware and benign files respectively.

Table 4.1: List of the top ranked API calls

ID	API Contributing to Malware Classification	Rel. Score
178	KERNEL32.DLL,VirtualQuery;	0.0568
124	KERNEL32.DLL,ExitProcess;	0.0459
615	KERNEL32.DLL,CreateFileW;	0.0406
607	KERNEL32.DLL,CompareStringA;	0.0381
8	USER32.DLL,RegisterClassA;	0.0355
1637	USER32.DLL,DestroyIcon;	0.0318
1606	USER32.DLL,TrackPopupMenu;	0.0317
207	KERNEL32.DLL,IsBadCodePtr;	0.0235
1601	USER32.DLL,CreatePopupMenu;	0.0213
235	USER32.DLL,DestroyWindow;	0.0205
ID	API Contributing to Benign File Classification	Rel. Score
80	KERNEL32.DLL,FreeLibrary;	0.1035
57	ADVAPI32.DLL,RegCloseKey;	0.0972
578	ADVAPI32.DLL,RegOpenKeyExW;	0.0964
20	KERNEL32.DLL,lstrlenW;	0.0846
111	KERNEL32.DLL,GetCurrentThreadId;	0.0825
22	KERNEL32.DLL,Sleep;	0.0756
37	KERNEL32.DLL,LocalFree;	0.0756
102	KERNEL32.DLL,GetTickCount;	0.0673
36	KERNEL32.DLL,GetLastError;	0.0538
506	KERNEL32.DLL,SetEvent;	0.0532

The most important activity in malware is file management [9], which enables them to create, or copy files (themselves or other files) multiple times to spread malware distribution, control the targeted computers, and destroy the integrity of the system (e.g., *CreateFileW* in *KERNEL32.DLL*, *DestroyIcon* in *USER32.DLL*, *CreatePopupMenu* in *USER32.DLL*, *DestroyWindow* in *USER32.DLL*, etc.). To achieve the malicious goals, they also have their own methods to deal with process and registry, which heavily use *VirtualQuery* in *KERNEL32.DLL* to get the virtual address space of the calling process that is intent to hide from or affect. Compared with malware, benign files act normally

in file, memory, process, and registry operations.

Based on the general statistical properties observed from the real sample collection, intuitively, to evade the detection with lower adversarial cost, the attackers may manipulate the API calls by the way of injecting the ones most relevant to benign files while removing the ones with higher relevance scores to malware. To stimulate the attacks, we rank each API call and group them into two sets: $\mathcal{M}$ (i.e., API calls highly relevant to malware) and $\mathcal{B}$ (i.e., API calls highly relevant to benign files) in the descent order of $I(x, +1)$ and $I(x, -1)$ respectively, where $\mathcal{M}$ is utilized for elimination, while $\mathcal{B}$ is applied for addition. It is worth noting that the observed sample files are either surrogate or originally used by the target system depending on different attack scenarios.

Adversarial Attack Model

To implement the adversarial attack, we further define a function $g(\mathcal{A}(\mathbf{X}))$ to represent the capability of an attacker:

$$g(\mathcal{A}(\mathbf{X})) = ||\mathbf{y} - \mathbf{f}'||^2, \tag{4.8}$$

where $\mathbf{f}' = \text{sign}(f(\mathcal{A}(\mathbf{X})))$, and $g(\mathcal{A}(\mathbf{X}))$ implies the number of malware misclassified as benign files. The underlying idea is thus to manipulate a subset of features with minimum adversarial cost while maximize the total loss of classification (as specified in Equation 4.8). In principle, a brute-force method can be applied to select features for manipulation. However, search by exhaustion is extremely expensive for the large-dimensional feature set. To achieve the optimal attack, here we adopt the wrapper method [152] which greedily selects features based on the capability of the attack. Different from the work in [152], we conduct bi-directional feature selection, that is, forward feature addition performed on $\mathcal{B}$ and backward feature elimination performed on $\mathcal{M}$. At each iteration, an API call will be selected for addition or elimination depending on the fact how it influences the value of $g(\mathcal{A}(\mathbf{X}))$. The adversarial attack $\boldsymbol{\theta} = \{\boldsymbol{\theta}^+, \boldsymbol{\theta}^-\}$ will be drawn from the iterations, where $\boldsymbol{\theta}^+ \in \{0,1\}^d$ (if API_i is selected for elimination, then $\theta_i^+ = 1$; otherwise, $\theta_i^+ = 0$), and $\boldsymbol{\theta}^- \in \{0,1\}^d$ (if API_i is selected for addition, then $\theta_i^- = 1$; otherwise, $\theta_i^- = 0$). The iterations will terminate at the point where the adversarial cost reaches to maximum ($\delta_{\max}$) or the features available for addition and elimination are all manipulated. The implementation of the proposed adversarial attack (*AdvAttack*) is given in Algorithm 4.

The proposed *AdvAttack* enables the adversary to fully take advantage of the property of the feature set, and get a better chance of evading the targeted classifier. $\mathcal{M}$

and $\mathcal{B}$ significantly decrease the number of searches, and thereby reduce the computational complexity. Given $m = \max(|\mathcal{M}|, |\mathcal{B}|)$, the proposed attack *AdvAttack* requires $O(n_t m(\mu^+ + \mu^-))$ queries, in which n_t is the number of testing malware samples, μ^+ and μ^- are the numbers of selected features for elimination and addition respectively. Note that, this algorithm is applicable to the attackers of different skills and capabilities resting on the feature space, the training data set, and the learning algorithm either surrogate or originally used by the targeted system.

Algorithm 4: ***AdvAttack*** - A well-crafted adversarial attack model for the attackers with different skills and capabilities

Input: Training set $D = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$, testing set $D_t = \{\mathbf{x}_i, y_i\}_{i=1}^{n_t}$; a_f: cost of changing feature f; c^+, c^-: cost of eliminating and adding features in a file; $\mathcal{S}^+$, $\mathcal{S}^-$: features selected; μ^+, μ^-: number of features selected for elimination and addition.

Output: Adversarial attack $\boldsymbol{\theta} = \{\boldsymbol{\theta}^+, \boldsymbol{\theta}^-\}$.

```
Train a classifier f(X) using n training file samples;
S+ ← ∅, S− ← ∅, θ+_i = 0, θ−_j = 0, (i, j ∈ (0, 1, ..., d));
while (c+ + c− ≤ δ_max) and (μ+ < d or μ− < d) do
    X ← X/S+, X ← X ⋃ S−;
    for each feature x+_i ∈ M do
        X_{x+} ← X/{x+_i}: eliminate x+_i from n_t testing file samples;
        Calculate g(X_{x+});
    end
    for each feature x−_j ∈ B do
        X_{x−} ← X ⋃ {x−_j}: add x−_j to n_t testing file samples; Calculate g(X_{x−});
    end
    x_max = argmax {g(X_{x+}), g(X_{x−})};
    if x_max ∈ M and μ+ < d then
        S+ ← S+ ⋃ {x+_i}
        c+ = c+ + a_{x+_i}, μ+ = μ+ + 1;
    end
    if x_max ∈ B and μ− < d then
        S− ← S− ⋃ {x−_j}
        c− = c− + a_{x−_j}, μ− = μ− + 1;
    end
end
Set θ+_i = 1 for x+_i ∈ S+, θ−_j = 1 for x−_j ∈ S−;
return θ = {θ+, θ−};
```

4.3.3 Secure-learning Model based on *AdvAttack*

A defender usually reacts to the adversarial attacks by analyzing the attack and retraining the classifier on the new collected file samples, or modifying features of the training dataset to counter the adversary's strategy [108]. However, retraining with adversarial data typically suffers from a limitation: the retrained model modifies the training data distribution approximate to the testing space through the attack model. After modifying a large number of features and malicious files, the model tends to produce a distribution that is very close to that of the benign files. In this case, the retrained model may not be able to differentiate benign and malicious files accurately. To this end, we perform our security analysis of the learning-based classifier resting on the application setting that the defender draws the well-crafted *AdvAttack* from the observed sample space, since the attack is modeled as optimization under generic framework. Therefore, in our proposed secure-learning model (*SecDefender*), we exploit the *AdvAttack* $\boldsymbol{\theta}$ to retrain the classifier in a progressive way and apply adversarial cost $\mathbf{c}$ to regularize the optimization problem.

Classifier Retraining. Incorporating the adversarial attack $\boldsymbol{\theta}$ into the learning algorithm can enables us to provide a significant connection between training and the adversarial action. Instead of manipulating the feature spaces for all the malicious training dataset, we start with the original training data $\mathbf{X}$ and iteratively computing a classifier by injecting the adversarial samples tainted by $\boldsymbol{\theta}$ into the training data that evade the previously computed classifier [83]. The new dataset $\mathbf{X}'$ can be formalized as follows:

$$\mathbf{X}' = \mathbf{X} \bigcup_{i=1}^{n_m} (\mathbf{x}_i + \boldsymbol{\theta}), \tag{4.9}$$

$$s.t.\ f(\mathbf{x}_i + \boldsymbol{\theta}) < 0, \tag{4.10}$$

where n_m is the total number of malware samples added during the retraining iteration. The iterations converge when there are no new adversarial samples generated through the retrained classifier or the specified number of iterations reaches. Compared to updating all the malicious training dataset, this progressive classifier retraining method effectively increases the importance of malware in training process, and can therefore significantly keep the detection system in a more accurate level.

Security Regularization. Resting on the retrained classifier, in our proposed model, we further enhance the security of the classifier by using a security regularization term over the adversarial cost. Our empirical studies demonstrate that even retrained

by the updated training dataset, the classifiers are still degraded to some extent. It's recalled that an optimal adversarial attack aims to manipulate a subset of features with minimum adversarial cost while maximize the total loss of classification. In contrast, to secure the classifier in malware detection, we would like to maximize the adversarial cost for the attacks [152]: from the analysis of the adversary problem [13, 78], we can find that the larger the adversarial cost, the more manipulations need to be performed, and the more difficult the attack is. If a larger number of features has to be manipulated to evade detection, it may be infeasible to perform such attack. Therefore, to be more resilient against the adversarial attack, an ideal secure-learning model is to maximize the adversarial cost for the attackers. Accordingly, the adversary action of the learning classifier can be defined as:

$$\mathcal{T}(\mathcal{A}(\mathbf{x}), \mathbf{x}) = \frac{1}{\mathcal{C}(\mathcal{A}(\mathbf{x}), \mathbf{x})}, \tag{4.11}$$

subject to Equation 4.3. If $\mathcal{A}(\mathbf{x}_i) = \mathbf{x}_i$ which represents that the file is not manipulated by the adversary, $\mathcal{T}(\mathcal{A}(\mathbf{x}_i), \mathbf{x}_i) = 0$. We then define an adversary action matrix denoted as $\mathbf{T} \in \mathbb{R}^{n \times n}$, where the element $T_{ij} = \mathcal{T}(\mathcal{A}(\mathbf{x}_i), \mathbf{x}_j)$. Based on the adjacency adversary action matrix $\mathbf{T}$, and the idea drawn from the Laplacian matrix [5], the security matrix can be defined as $\mathbf{S} = \mathbf{D} - \mathbf{T}$, where $\mathbf{D}$ is the diagonal matrix with $\mathbf{D}_{ii} = \sum_k \mathbf{T}_{ik}$ while the remain elements are 0. Resting on the concept of label smoothness [138] and assumption for the optimization learning [110] (i.e., data points tend to reserve the initial labels over the classification), we can secure the classifier with the constraint as $\frac{1}{2}\sum_{i,j=1}^{n} T_{ij}(f'_i - y_j)^2 = \frac{1}{2}\mathbf{f}'^T\mathbf{S}\mathbf{y}$.

Since the learning-based malware detection can be formalized as an optimization problem denoted by Equation 4.2, we can then bring a regularization term to enhance its security. This constraint penalizes parameter choices, smooths the effects the attack may cause, and in turn helps to promote the optimal solution for the local minima in the optimization problem. Therefore, to minimize classifier sensitivity to feature manipulation, we can minimize the security regularization term. Based on Equation 4.2, we can formulate a secure-learning model against the adversarial attack as:

$$\begin{aligned} \underset{\mathbf{f}', \mathbf{w}, \mathbf{b}; \boldsymbol{\xi}}{\operatorname{argmin}} \; \mathcal{L}(\mathbf{f}', \mathbf{w}, \mathbf{b}; \boldsymbol{\xi}) = \underset{\mathbf{f}', \mathbf{w}, \mathbf{b}; \boldsymbol{\xi}}{\operatorname{argmin}} \; & \frac{1}{2}||\mathbf{y} - \mathbf{f}'||^2 + \frac{\alpha}{2}\mathbf{f}'^T\mathbf{S}\mathbf{y} + \\ & \frac{1}{2\beta}\mathbf{w}^T\mathbf{w} + \frac{1}{2\gamma}\mathbf{b}^T\mathbf{b} + \boldsymbol{\xi}^T(\mathbf{f}' - \mathbf{X}'^T\mathbf{w} - \mathbf{b}). \end{aligned} \tag{4.12}$$

where α is the regularization parameter for the security constraint. As $\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0$, $\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = 0$,

$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}} = 0$, $\frac{\partial \mathcal{L}}{\partial \mathbf{f}'} = 0$, we have

$$\mathbf{w} = \beta \mathbf{X}' \boldsymbol{\xi}, \tag{4.13}$$

$$\mathbf{h} = \gamma \boldsymbol{\xi}, \tag{4.14}$$

$$\mathbf{f}' = \mathbf{X}'^T \mathbf{w} + \mathbf{h}, \tag{4.15}$$

$$\mathbf{f}' = \mathbf{y} - \frac{\alpha}{2} \mathbf{S} \mathbf{y} - \boldsymbol{\xi}. \tag{4.16}$$

Based on the derivation from Equation 4.13, Equation 4.14, and Equation 4.15, we have

$$\boldsymbol{\xi} = (\beta \mathbf{X}'^T \mathbf{X}' + \gamma \mathbf{I})^{-1} \mathbf{f}'. \tag{4.17}$$

We substitute Equation 4.17 to Equation 4.16, then we get the final secure-learning problem as:

$$((\beta \mathbf{X}'^T \mathbf{X}' + \gamma \mathbf{I}) + \mathbf{I}) \mathbf{f}' = (\mathbf{I} - \frac{\alpha}{2} \mathbf{S})(\beta \mathbf{X}'^T \mathbf{X}' + \gamma \mathbf{I}) \mathbf{y}. \tag{4.18}$$

Since the size of $\mathbf{X}'$ is $d \times n$, the computational complexity for Equation 4.18 is $O(n^3)$. To solve the secure-learning problem (Equation 4.18), we use conjugate gradient descent method and the implementation of *SecDefender* is shown in Algorithm 5.

Algorithm 5: *SecDefender* - A secure-learning model against well-crafted attack

---

Input: Training data set $D = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ and testing set $D_t = \{\mathbf{x}_i, y_i\}_{i=1}^{n_t}$; Evasion attack $\boldsymbol{\theta}$.

Output: $\mathbf{f}'$: the labels of the input files.

Iteratively train classifier $f(\mathbf{X} \bigcup_i (\mathbf{x}_i + \boldsymbol{\theta}))$ to get $\mathbf{X}'$;

$\mathbf{f}'_0 = 0$;

$\mathbf{A} = (\beta \mathbf{X}'^T \mathbf{X}' + \gamma \mathbf{I}) + \mathbf{I}$;

$\mathbf{r}_0 = (\mathbf{I} - \frac{\alpha}{2} \mathbf{S})(\beta \mathbf{X}'^T \mathbf{X}' + \gamma \mathbf{I}) \mathbf{y} - \mathbf{A} \mathbf{f}'_0$;

$\mathbf{p}_0 = \mathbf{r}_0$;

$k = 0$;

while $||\mathbf{r}_k|| > \varepsilon$ **do**

- $\lambda_k = (\mathbf{r}_k^T \mathbf{r}_k)/(\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k)$;
- $\mathbf{f}'_{k+1} = \mathbf{f}'_k + \lambda_k \mathbf{p}_k$;
- $\mathbf{r}_{k+1} = \mathbf{r}_k - \lambda_k \mathbf{A} \mathbf{p}_k$;
- $\zeta_k = (\mathbf{r}_{k+1}^T \mathbf{r}_{k+1})/(\mathbf{r}_k^T \mathbf{r}_k)$;
- $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \zeta_k \mathbf{p}_k$;
- $k = k + 1$;

end

return $\mathbf{f}'_k$;

4.3.4 Experimental Results and Analysis

In this section, to empirically validate the proposed secure-learning model *SecDefender*, we present four sets of experiments: (1) In the first set of experiments, we compare the attack model *AdvAttack* with other feature manipulation methods; (2) In the second set of experiments, we evaluate the attack model *AdvAttack* under different scenarios; (3) In the second set of experiments, we then evaluate the effectiveness of our proposed secure-learning model *SecDefender* against the adversarial attack; (4) In the last set of experiments, we compare the performance of *SecDefender* against the adversarial attack with other widely used anti-malware products. We use the same performance indices shown in Table 3.5.

Experimental Setup

The real sample collection obtained from Comodo Cloud Security Center contains $10,000$ file samples with $3,503$ extracted API calls, where $5,000$ are malware, $5,000$ are benign files. In our experiments, we randomly select 90% of the samples for training, while the remaining 10% is used for testing. Since not all of the API calls will contribute to the classification as analyzed in Section 4.3.2, those API calls whose relevance scores are lower than the empirical threshold (i.e., 0.0005 in our application) will be excluded for feature manipulations. Therefore, $|\mathcal{M}| = 810$, $|\mathcal{B}| = 1,183$, and all the file samples can be represented as binary feature vectors with $1,993$-dimensions. For simplicity, we assume $c_i = 1$ for each feature to conduct our experiments.

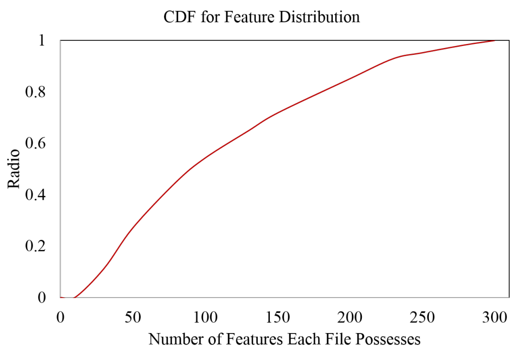


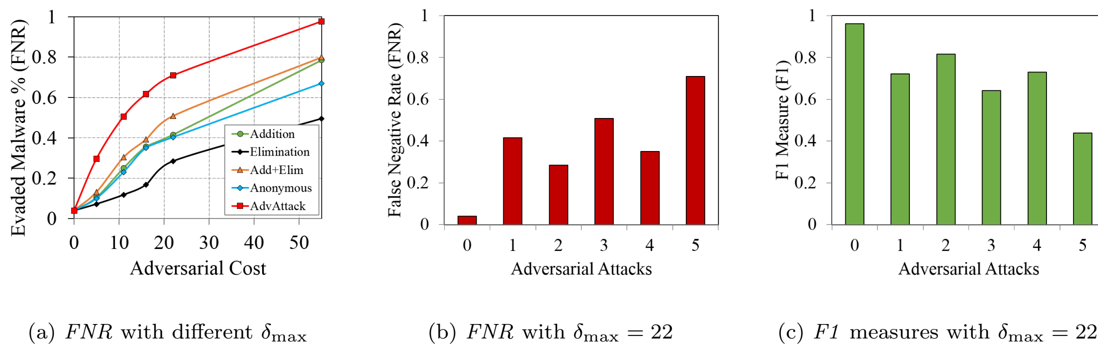
Figure 4.5: The feature distribution of file samples.

According to the Cumulative Distribution Function (CDF) for the number of API calls the file samples include shown in Figure 4.5, we exploit the average number of API calls that each file possesses, which is 109, to define the maximum manipulation

cost $\delta_{\max}$. We run our evaluation of the proposed adversarial attacks with $\delta_{\max}$ varies in {5%, 10%, 15%, 20%, 50%} of 109, which is {5, 11, 16, 22, 55}. We also use the performance measures shown in Table 3.5 to quantitatively validate the effectiveness of the proposed methods.

Comparisons of *AdvAttack* and Other Attacks

We first compare our proposed adversarial attack *AdvAttack* with other attack methods using different feature manipulation approaches including: (1) only manipulating API calls from $\mathcal{B}$ for addition; (2) only manipulating API calls from $\mathcal{M}$ for elimination; (3) sequentially selecting $(1/2 \times \delta_{\max})$ API calls from $\mathcal{B}$ for addition and $(1/2 \times \delta_{\max})$ API calls from $\mathcal{M}$ for elimination; (4) simulating anonymous attack by randomly manipulating API calls for addition and elimination.



(a) *FNR* with different $\delta_{\max}$ (b) *FNR* with $\delta_{\max} = 22$ (c) *F1* measures with $\delta_{\max} = 22$

Figure 4.6: Comparisons of *AdvAttack* and other adversarial attacks: *Original-Classifier* (0), different adversarial attacks (Method 1 - 4) and *AdvAttack* (5)

The experimental results are shown in Figure 4.6. Note that, Since we just manipulate the features on the testing malicious files (i.e., benign files remain unchanged), all *FP*s and *TN*s after attacks in the experiments keep the same as before attack (i.e., *FP* is 21, and *TN* is 423). The experimental results illustrate that the attack performances vary when using different feature manipulation methods with certain adversarial costs $\delta_{\max}$: (1) the manipulation of only feature elimination performs worst with *FNR*; (2) the manipulation which sequentially selecting features for addition and elimination performs better than the methods only using feature addition or elimination, and the anonymous attack, due to its bi-directional feature manipulation over $\mathcal{B}$ and $\mathcal{M}$; (3) *AdvAttack* can greatly improve the *FNR* to 0.6978 while degrade the detection *F1* measure of the classifier to 0.4384, when $\delta_{\max} = 22$; the attackers can achieve ideal attack using *AdvAttack* (i.e., *FNR* almost reaches to 1, which means almost malware samples are misclassified),

when $\delta_{\max} = 55$. Due to its well-crafted attack strategy, *AdvAttack* outperforms other adversarial attack methods with different feature manipulation approaches.

Evaluations of *AdvAttack* under Different Scenarios

We further implement and evaluate our proposed attack *AdvAttack* under different scenarios described in Section 4.2: (1) In mimicry (MMC) attack ($\Psi = (X, \hat{D})$), the attackers are assumed to know the feature space and be able to obtain a file collection to imitate the original training dataset. In our experiment, we randomly select $1,000$ file samples (500 benign and 500 malicious) from the $9,000$ training set as our mimic dataset and exploit commonly used linear SVM as the surrogate classifier to train these $1,000$ mimic file samples. (2) In imperfect-knowledge (IPK) attack ($\Psi = (X, D)$), we assume that both the feature space and the original training sample set can be fully controlled by the attackers. Therefore, we perform the IPK attack conformably as MMC attack where the only difference is that we apply $9,000$ samples to train SVM. (3) In Ideal-knowledge (IDK) attack ($\Psi = (X, D, f)$), the attackers can perfectly access to the classifier system. The previous experiments of *AdvAttack* are conducted based on such assumption. To be comparable, *AdvAttack* is applied to all these scenarios resting on the same cost settings.

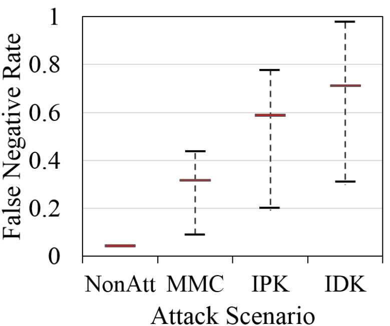


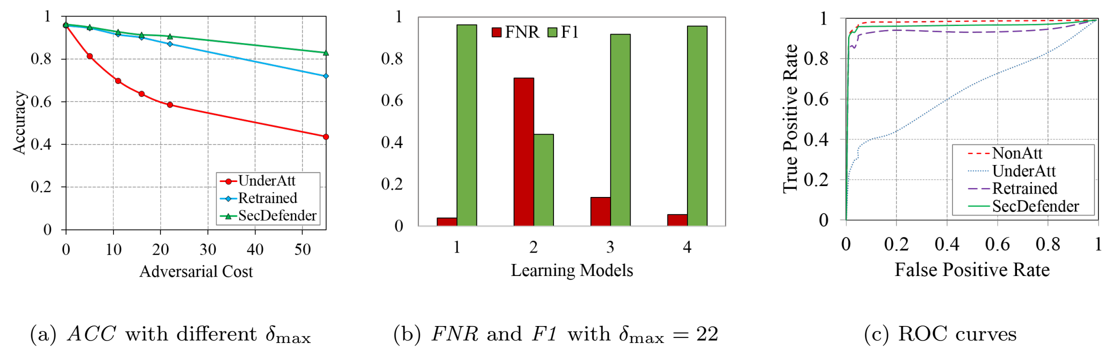
Figure 4.7: *FNR* before and after each attack under different scenarios for all 1,000 testing file samples

The experimental results are shown in Figure 4.7, in which red bar denotes the *FNR* of the classifier before attack (NonAtt) and different attacks with the adversarial cost $\delta_{\max} = 22$. The *FNR* values float up or down depending on the adversarial cost. The experimental results demonstrate that the available knowledge for the attackers significantly contributes to the performance of the attack. With perfect knowledge, the IDK attack can well evade the detection (e.g., 69.78% of the testing malware samples are misclassified as benign when $\delta_{\max} = 22$). Our proposed attack *AdvAttack* can be

applied as a representative attack model with general attack characteristics.

Evaluation of *SecDefender* against Adversarial Attacks

In response to well-crafted attacks, we'd like to assess the effectiveness of our proposed secure-learning model *SecDefender* based on *AdvAttack*. We use *AdvAttack* to taint the malware in the testing sample set, and validate the classification performance in different ways: (1) the *Original-Classifier* before attack (*NonAtt*); (2) the classifier under attack (UnderAtt); (3) the classifier retrained using the updated training dataset (i.e., $\mathbf{x}+\boldsymbol{\theta}$) (Retrained); (4) our secure-learning model *SecDefender*. The comparisons of the effectiveness of these classifiers are shown in Figure 4.8(a) (accuracy values against attacks with different adversarial costs) and Figure 4.8(b),(c) (FNR, F1, and ROC curves for the classifiers against the attack with $\delta_{\max} = 22$).



(a) *ACC* with different $\delta_{\max}$ (b) *FNR* and *F1* with $\delta_{\max} = 22$ (c) ROC curves

Figure 4.8: Comparisons of *SecDefender* and other classification models on *ACC*, *F1*, *FNR*, and ROC curves: *Original-Classifier* (1), *Original-Classifier* under attack (2), retrained *Original-Classifier* (3), and *SecDefender* (4)

It can be observed that the retrained classifier ideally applying the adversarial attack $\boldsymbol{\theta}$ to transform the malware in the training dataset from $\mathbf{x}$ to $\mathbf{x} + \boldsymbol{\theta}$ can somehow be resilient to the attacks, but the accuracy still remain unsatisfied. In contrast, *SecDefender* with progressive retraining technique, can well improve the *TPR* and accuracy, and bring the malware detection system back up to the desired performance level, the detection *F1* measure of which is 0.9561 ($\delta_{\max} = 22$), approaching the detection results before the attack (i.e., 0.9613).

It may also be interesting to know how robust that our learning systems can combat the anonymous attacks. We conduct the anonymous attack by randomly selecting the features for addition or elimination as described in Experiment 4.3.4, which does not exploit any knowledge of the target system. Under the anonymous attack, *SecDefender*

has zero knowledge of what the attack is. Even in such case, *SecDefender* still improves the detection *F1* measure from 0.7304 to 0.8830. Based on these properties, *SecDefender* can be a resilient solution in malware detection against well-crafted attacks even the attackers have perfect knowledge of the learning system.

Comparisons with Different Anti-malware Scanners

In this set of experiments, we evaluate the performance of *SecDefender* against the adversarial attack in comparison with some other popular commercial anti-malware scanners such as Kaspersky (K), McAfee (M), Symantec (S), and TrendMicro (T). For the comparisons, we use all the latest versions of the security products. We use 556 malware samples from the testing dataset described in Section 4.3.4 for evaluation. The testing malware are first tainted by *AdvAttack*, and then scanned by these anti-malware products. The detection results are illustrated in Table 4.2. Compared with these typical anti-malware scanners, *SecDefender* can effectively sustain the *TPR* to 0.9335, and performs the best accurate detection.

Table 4.2: Comparisons of different anti-malware scanners

Malware	K	M	S	T	*SecDefender*
1	×	√	×	×	√
2	√	×	√	×	×
3	×	×	×	×	√
4	×	√	×	×	√
5	×	×	×	×	√
6	×	×	×	√	√
7	×	×	×	×	√
8	×	×	×	×	×
9	√	×	√	×	√
10	√	×	×	×	√
⋮	⋮	⋮	⋮	⋮	⋮
556	×	√	√	√	√
TP	508	503	511	498	519
TPR	0.9136	0.9046	0.9190	0.8957	0.9335

4.4 ***SecureDroid***: A Secure-learning Paradigm against Various Kinds of Attacks

Though *SecDefender* is promising, it makes strong assumptions about the structure of the data (e.g., adversarial samples) and the attack model that are likely impractical for malware detection problems. The effectiveness for these methods depends on the adversarial attacks similar to the one used by the adversary, which is non-adaptive to the unknown attacks. In this section, we aim to enhance security of machine learning-based malware detection against various kinds of adversarial attacks, whose action space is practically independent from the skills and capabilities of the attackers.

According to Lemma 4.1, in the adversarial point of view, to conduct a practical attack, attackers intend to find the features which are easy to be manipulated (i.e., features with low costs being manipulated) and minimize the manipulations (i.e., modify the features as less as possible) to bypass the detection. For example, to evade the detection, attackers may manipulate the spy Trojan *"Trojan-Spy.Win32.Zbot"* by injecting the Windows API calls of *"KERNEL32.DLL,FreeLibrary;"* which is frequently used in benign files instead of removing suspicious API call of "*MAPI32.MAPIReadMail*", since feature addition is usually cost-effective and safer than feature elimination to bypass the detection while preserves the semantics and intrusive functionality of the original malicious file. In contrast, to be resilient against the adversarial attacks, an ideal defense should make the attackers cost-expensive and maximize their manipulations to evade the detection.

In this section, resting on the analysis of a set of features (i.e., permissions, filtered intents, API calls, and new-instances) extracted from the Android app files, we take a further step to explore the security of machine learning in malware detection. To make the classifier harder to be evaded, we first present a novel feature selection method (named *SecCLS*) to build the classifier, by taking consideration of different importances of the features associated with their contributions to the classification problem as well as their manipulation costs. To improve the system security while not compromising the detection accuracy, we further propose an ensemble learning approach (named *SecENS*) by aggregating the individual classifiers that are constructed using the proposed feature selection method *SecCLS*. Accordingly, we develop a system called *SecureDroid* which integrates both *SecCLS* and *SecENS* to secure machine learning-based malware detection. The system architecture of *SecureDroid* is shown in Figure 4.9.

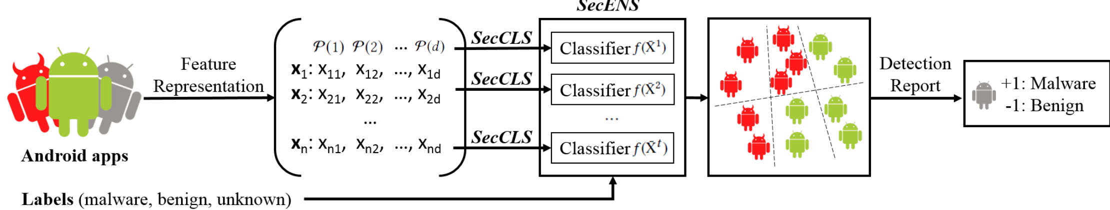


Figure 4.9: An overview of system architecture of *SecureDroid*. In the system, the collected app files are first represented as d-dimensional binary feature vectors. Then *SecCLS* is applied to select a set of features (each feature i is selected with probability $\mathcal{P}(i)$) to construct a more secure classifier. *SecENS* is later exploited to aggregate different individual classifiers built using *SecCLS* to classify malicious and benign files. For a new file, based on the extracted features, it will be predicted as either malicious or benign based on the trained classification model.

4.4.1 Feature Representation

Since we use Android apps as a case study to investigate the secure-learning paradigm *SecureDroid*, we would like to introduce feature representations of Android apps with preliminaries. Unlike traditional PE file, Android app is compiled and packaged in a single archive file (with an .apk suffix) that contains the manifest file, Dalvid executable (dex) file, resources, and assets.

Manifest file. Android defines a component-based framework for developing mobile apps, which is composed of four different types of components [67]: *Activities* provide Graphical User Interface (GUI) functionality to enable user interactivity; *Services* are background communication processes that pass messages between the components of the app and communicate with other apps; *Broadcast Receivers* are background processes that respond to system-wide broadcast messages as necessary; and *Content Providers* act as database management systems that manage the app data. Android app must declare its components in the manifest file which retains information about its structure. Before the Android system can start an app component, the system must know that the component exists by reading the app's manifest file. The manifest file actually works as a road map to ensure that each app can function properly in the Android system. The actions of each component are further specified through *filtered intents* which declare the types of intents that an activity, service, or broadcast receiver can respond to [3]. For example, through filtered intents, an activity can initiate a phone call or a broadcast receiver can monitor SMS message. The manifest file also contains a list of *permissions*

requested by the app to perform functions (e.g., access Internet). Since permissions and filtered intents can reflect the interaction between an app and other apps or operation system, we extract them from manifest file as features to represent Android apps.

Dalvid executable (dex). Android apps are usually developed with Java. Development environments (e.g., Eclipse) convert the Java source codes into Dalvik executable (dex) files which can be run on the Dalvik Virtual Machine (DalvikVM)[1] in Android. Dex is a file format that contains compiled code written for Android and can be interpreted by the DalvikVM, which includes all the user-implemented methods and classes. Dex file always contains *API calls* that are used by the Android apps in order to access operating system functionality and resources, and *new-instances* which can be used to create new instances of classes from operating system classes. Therefore, both API calls and new-instances in the dex file can be used to represent the behaviors of an Android app. To extract them from a dex file, since dex file is unreadable, we (1) first use the reverse engineering tool APKTool[2] to decompile the dex file into smali code (i.e., the intermediate but interpreted code between Java and DalvikVM); and (2) then parse the converted smali code to extract these two kinds of features.

We refer the readers to Section 4.3 for a statement on why we perform static analysis and what the scope of our work is. The above Android apps and their features are exploited as a case study which facilitate the understanding of our further proposed approach, while other types of files and feature extractions are also applicable in our further investigation.

Similar to the feature representation in Section 4.3.1, to represent each collected Android app, we first extract the features and convert them into a vector space, which can be fed to the classifier either for training or testing. For the collected apps, we extract four sets of features (*S1* – *S4*) to represent them (shown in Table 4.3): permissions (*S1*) and filtered intents (*S2*) from manifest files, API calls (*S3*) and new-instances (*S4*) from dex files.

Resting on the above extracted features, we denote our dataset D to be of the form $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$ of n apps, where $\mathbf{x}_i$ is the features extracted from app i, and y_i is the class label of app i ($y_i \in \{+1, -1, 0\}$, $+1$: malicious, -1: benign, and 0: unknown). Let d be the number of all extracted features in *S1* – *S4* in dataset D. Each app can then

[1]https://source.android.com/devices/tech/dalvik/.
[2]http://ibotpeaches.github.io/Apktool/

Table 4.3: Illustration of extracted features for Android apps in *SecureDroid*

	Features	Examples
Manifest	*S1:* Permissions	*READ_PHONE_STATE* *INTERNET*
	S2: Filtered Intents	*intent.action.MAIN* *vending.INSALL_REFERER*
Dex	*S3:* API calls	*getSimSerialNumber* *containsHeader*
	S4: New-Instances	*Ljave/util/HashMap* *Landroid/app/ProgressDialog*

be represented by a binary feature vector:

$$\mathbf{x}_i = \begin{pmatrix} 0 \\ \dots \\ 1 \\ 1 \\ \dots \\ 0 \\ 1 \\ \dots \\ 0 \\ 1 \\ \dots \\ 1 \end{pmatrix} \rightarrow \begin{array}{l} \text{READ_PHONE_STATE} \\ \dots \\ \text{INTERNET} \\ \text{intent.action.MAIN} \\ \dots \\ \text{vending.INSALL_REFERER} \\ \text{getSimSerialNumber} \\ \dots \\ \text{containsHeader} \\ \text{Ljave/util/HashMap} \\ \dots \\ \text{Landroid/app/ProgressDialog} \end{array} \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \textit{S1: } \text{Permissions} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \textit{S2: } \text{Filtered Intents} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \textit{S3: } \text{API calls} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \textit{S4: } \text{New-Instances} \end{array}$$

where $\mathbf{x}_i \in \mathbb{R}^d$, and $x_{ij} = \{0, 1\}$ (i.e., if app i includes feature j, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$).

4.4.2 Secure Classifier Construction using Novel feature Selection

In adversarial settings, the importance of a feature from an attacker's perspective is: (i) its contribution to the classification problem, which is corresponding to the weight w trained by the classifier based on the training data, and (ii) its complexity being manipulated, that is, the manipulation cost c decided by feature type (e.g., permission

vs. API call) and manipulation method (e.g., addition vs. elimination). Given ith-feature ($1 \leq i \leq d$) extracted from the dataset D, its importance can be defined as follow:

$$\mathcal{I}(i) \propto \frac{|w_i|}{c_i}, \tag{4.19}$$

which implies that the larger the weight of the feature trained by the classifier and the lower the cost of the feature being manipulated, the more important the feature to the attackers. Clearly, the importance of a feature represents the possibility that an attacker may manipulate it in an adversarial attack.

The rationale to construct a more secure classifier against the adversarial attacks is to reduce the possibility of those important features being selected for model construction. In other words, those features the attackers tend to manipulate (i.e., features with higher values of $\mathcal{I}(i)$) may not present together in the learning model, which will intuitively force attackers to manipulate a larger number of other less important features (i.e., features with lower values of $\mathcal{I}(i)$) to evade the detection. In this way, the probability of each feature being selected for constructing a classification model is inversely proportional to its importance, that is, the more important the feature is to attackers, the less possible it will be selected to train the classifier. We formalize $\mathcal{P}(i)$, the probability of ith-feature being selected, as:

$$\mathcal{P}(i) \propto \frac{\lambda}{\mathcal{I}(i)}, \tag{4.20}$$

where λ is an adjustable parameter which can be empirically decided based on the training data. When substituting Eq. (4.19) into Eq. (4.20), the length of the probability is actually arbitrary long (e.g., $|w_i| = 0$). To normalize $\mathcal{P}(i)$, we further define $\mathcal{P}(i)$ as:

$$\mathcal{P}(i) = \lambda c_i (1 - \rho |w_i|), \tag{4.21}$$

where ρ ($0 < \rho < 1$) is a rescaling parameter to keep $\mathcal{P}(i)$ in the range of $(0, 1]$.

For the weight w_i of ith-feature, it can be calculated by the learning-based classifier in Eq. (4.2) trained on the dataset D. Provided that Eq. (4.2) is an optimization problem, based on the derivation and substitution, the weight vector for all features can be calculated as:

$$\mathbf{w} = \beta \mathbf{X} \boldsymbol{\xi}, \tag{4.22}$$

$$s.t.\ \boldsymbol{\xi} = (\beta \mathbf{X}^T \mathbf{X} + \gamma \mathbf{I})^{-1} \mathbf{f}, \tag{4.23}$$

where $\mathbf{f}$ can be solved through Eq. (4.2) using conjugate gradient descent method. We then further normalize each weight $|w_i|$ using *min-max normalization* [59] to the range of $[0, 1]$.

For the manipulation cost c_i of ith-feature, as discussed in Section 4.2.2, it can be estimated with respect to its feature type and the manipulation method. Considering that (1) feature addition is usually easier than elimination, and (2) compared with permissions and filtered intents in the manifest file, API calls and new-instances in dex file are relatively easier to be manipulated, Figure 4.10 illustrates different costs empirically decided for manipulating different kinds of features in our application.

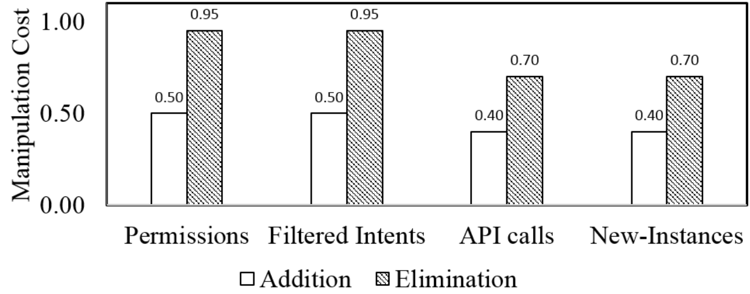


Figure 4.10: The manipulation costs determined by different feature types and manipulation methods.

Attacker may know completely, partially, or do not have any information of the targeted learning system about: (i) the feature space, (ii) the training data set, and (iii) the learning algorithm [126]. We would like to overestimate attackers' capabilities rather than underestimate them. Since this worst case provides a potential upper bound on the performance degradation suffered by the learning system under the adversarial attacks, it can be used as reference to evaluate the effectiveness of the learning system under the other limited attack scenarios. To conduct a well-crafted attack, we assume that attackers are capable to access the targeted learning system and may have perfect knowledge regarding the system. Therefore, they can use the methods such as *information gain* [59] or *max-relevance* [103] to calculate the information of each feature for the classification of malicious and benign apps respectively. Then they will be able to utilize those features that significantly contribute to benign apps classification for additions, and apply those ones that significantly contribute to malicious apps classification for eliminations.

As the above presentation, we can see that $\mathcal{P}(i) \in (0, 1]$, where the minimum is attained when the feature is most informative for the classification task or easy to be manipulated, and the maximum is attained when the feature has least contribution to the classification problem or is too costly to be manipulated. We then form the probability

set for selecting features to construct a more secure classifier as:

$$\mathcal{P} = \{\mathcal{P}(1), \mathcal{P}(2), ..., \mathcal{P}(d)\}. \tag{4.24}$$

Given a pseudo random function $\mathcal{R}(.) \in (0,1)$, the original feature vector of a given app $\mathbf{x}_i$ will be represented by an updated binary feature vector $\bar{\mathbf{x}}_i$:

$$\bar{\mathbf{x}}_i =< \bar{x}_{i1}, \bar{x}_{i2}, \bar{x}_{i3}, ..., \bar{x}_{id} >,$$

where

$$\bar{x}_{ij} = \begin{cases} x_{ij} & \mathcal{R}(.) \le \mathcal{P}(j) \\ 0 & \text{otherwise} \end{cases}. \tag{4.25}$$

The proposed feature selection method for classifier construction is named *SecCLS*, whose implementation is given in Algorithm 6. Note that when $\mathcal{P}(i)$ $(1 \le i \le d)$ is with the same value for each feature, i.e., feature importances are evenly distributed, our proposed feature selection method *SecCLS* is approximate to random selection. Thus we can say, random feature selection method [63, 16] is a lower bound of *SecCLS*. Our proposed feature selection method *SecCLS* reduces the possibility of those features that attackers tend to manipulate, which will accordingly force attackers to manipulate a larger number of other features and thus be more resilient against their attacks. For computational complexity of *SecCLS*, to get weight vector $\mathbf{w}$ from Eq. (4.2) requires $O(d^3)$ queries, while to form cost vector $\mathbf{c}$ and calculate $\mathcal{P}$ both need $O(d)$. Since we formalize n apps as $\mathbf{X}$, each column of which is the d-dimensional feature vector, to get an updated training set $\bar{\mathbf{X}}$ from $\mathbf{X}$ requires $O(nd)$ updates.

By using *SecCLS*, after feature selection, the learning-based classifier in Eq. (4.2) can be updated as:

$$\underset{\bar{\mathbf{f}},\mathbf{w},\mathbf{b};\boldsymbol{\xi}}{\operatorname{argmin}} \frac{1}{2}||\mathbf{y} - \bar{\mathbf{f}}||^2 + \frac{1}{2\beta}\mathbf{w}^T\mathbf{w} + \frac{1}{2\gamma}\mathbf{b}^T\mathbf{b} + \boldsymbol{\xi}^T(\bar{\mathbf{f}} - \bar{\mathbf{X}}^T\mathbf{w} - \mathbf{b}), \tag{4.26}$$

subject to $\bar{\mathbf{f}} = \text{sign}(f(\bar{\mathbf{X}}))$, where $\bar{\mathbf{f}}$ is the predicted label vector based on a feature set $\bar{\mathbf{X}}$. To solve the problem in Eq. (4.26), let

$$\mathcal{L}(\bar{\mathbf{f}}, \mathbf{w}, \mathbf{b}; \boldsymbol{\xi}) = \frac{1}{2}||\mathbf{y} - \bar{\mathbf{f}}||^2 + \frac{1}{2\beta}\mathbf{w}^T\mathbf{w} + \frac{1}{2\gamma}\mathbf{b}^T\mathbf{b} + \boldsymbol{\xi}^T(\bar{\mathbf{f}} - \bar{\mathbf{X}}^T\mathbf{w} - \mathbf{b}). \tag{4.27}$$

Based on the substitution and derivation from $\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = 0$, $\frac{\partial\mathcal{L}}{\partial\mathbf{b}} = 0$, $\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}} = 0$, $\frac{\partial\mathcal{L}}{\partial\bar{\mathbf{f}}} = 0$, we can get the more secure classifier as:

$$[\mathbf{I} + (\beta\bar{\mathbf{X}}^T\bar{\mathbf{X}} + \gamma\mathbf{I})^{-1}]\bar{\mathbf{f}} = \mathbf{y}. \tag{4.28}$$

Algorithm 6: ***SecCLS*** – A novel feature selection method to construct more secure classifier.

Input: Training data set $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$.

Output: $\bar{\mathbf{X}}$: updated training set based on the selected features.

```
Get weight vector w by the learning-based classifier in Eq. (4.2) trained on D;
Get manipulation cost vector c;
Calculate P = {P(1), P(2), ..., P(d)} using Eq. (4.21);
k = 1;
for k ≤ d do
    Get a pseudo random number from R(.);
    if R(.) ≤ P(k) then
        X̄_k. = X_k.
    else
        X̄_k. = 0
    end
    k++;
end
return X̄;
```

4.4.3 Ensemble Learning to Improve Detection Accuracy

In the previous section, a novel feature selection method *SecCLS* is presented for constructing a more secure classifier against the adversarial attacks. To improve the system security while not compromising the detection accuracy, in this section, we further propose an ensemble learning approach called *SecENS* to aggregate a set of classifiers built using *SecCLS* to generate the final output for the detection.

Ensemble methods are a popular way to overcome instability and increase performance in many machine learning tasks [147], such as classification, clustering and ranking. An *ensemble* of classifiers is a set of classifiers whose individual decisions are combined in some way (e.g., by weighted or unweighted voting) to classify new samples, which is shown to be much more accurate than the individual classifiers that make them up [141]. Typically, an ensemble can be decomposed into two cascaded components: the first component is to create base classifiers with necessary accuracy and diversity; the second one is to aggregate all of the outputs of base classifiers into a numeric value as the final output of the ensemble. In general, base classifiers are generated by subsampling training set or input features (as done in boosting or bagging), manipulating the output targets, or injecting randomness in the learning algorithm [43].

In this paper, with certain accuracy of each individual classifier, we aim to diversify the classifiers that form the ensemble while also consider the integration of whole feature space. More specifically, the set of classifiers in the ensemble should follow two criteria: (1) the feature set used for building each classifier should differentiate from each other (i.e., *feature differentiation*), and (2) the ensemble should cover as many features as possible to assure the integration of whole feature space (i.e., *feature integration*). Therefore,

---

Algorithm 7: *SecENS* – An ensemble learning approach to improve the detection accuracy.

Input: Training data set $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$; $\mathcal{W}$: weights of training files; ε: error rate for each classifier; ζ: importance of each classifier; η_d: specified threshold of f_D; η_f: specified threshold of f_I; η_a: specified threshold of training accuracy.

Output: $\mathbf{f}$: the labels for the files.

Initialize: $\mathcal{W}_1(i) = \frac{1}{n}$ for $i = 1, 2, ..., n$; $t = 0$;
Get training set $\bar{\mathbf{X}}^1$ and selected feature set $\mathbf{F}^1$ on D using *SecCLS*;
while *1* **do**
    $t++$;
    Train a base classifier using $\bar{\mathbf{X}}^t$;
    Get weak hypothesis f_t: $\bar{\mathbf{X}}^t \rightarrow \{-1, 1\}$;
    Calculate error rate of f_t:
    $\varepsilon_t \leftarrow \sum_{i=1}^n \mathcal{W}_t(i)[y_i \neq f_t(\bar{\mathbf{x}}_i^t)]$;
    Set $\zeta_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$;
    Update $\mathcal{W}_{t+1}(i) = \frac{\mathcal{W}_t(i)\exp(-\zeta_t \bar{y}_i f_t(\bar{\mathbf{x}}_i))}{\sum_{i=1}^n \mathcal{W}_t(i)\exp(-\zeta_t \bar{y}_i f_t(\bar{\mathbf{x}}_i))}$ for $i = 1, 2, ..., n$;
    Calculate $f_I(\mathbf{F})$;
    Calculate the training accuracy (acc) of the ensemble based on
     $\mathbf{f} = \text{sign}(\sum_{i=1}^t \zeta_i f_i(\mathbf{X}))$;
    if $f_I(\mathbf{F}) \geq \eta_f$ and $acc \geq \eta_a$ **then**
        break;
    end
    Get $\bar{\mathbf{X}}^{t+1}$ and $\mathbf{F}^{t+1}$ on D using *SecCLS*;
    Calculate $f_D(\mathbf{F}^{t+1}, \mathbf{F}^j)$ for $j = 1, 2, ..., t$;
    while $\min\{f_D(\mathbf{F}^{t+1}, \mathbf{F}^j) \mid j = 1, 2, ..., t\} < \eta_d$ **do**
        Get $\bar{\mathbf{X}}^{t+1}$ and $\mathbf{F}^{t+1}$ on D using *SecCLS*;
        Calculate $f_D(\mathbf{F}^{t+1}, \mathbf{F}^j)$ for $j = 1, 2, ..., t$;
    end
end
return $\mathbf{f} = \text{sign}(\sum_{i=1}^t \zeta_i f_i(\mathbf{X}))$;

---

we first construct each individual classifier using the proposed feature selection method *SecCLS* described in Section 4.4.2; then we follow the above two criteria and propose

SecENS to aggregate a set of the constructed classifiers to generate the final output for the detection. We present *SecENS* with the definitions of the above two criteria.

Feature Differentiation (denoted as f_D). Given two feature sets $\mathbf{F}^a \in \mathbb{R}^d$ and $\mathbf{F}^b \in \mathbb{R}^d$, which are selected using the proposed method *SecCLS* respectively, the differentiation between them can be defined as:

$$f_D(\mathbf{F}^a, \mathbf{F}^b) = 1 - J(\mathbf{F}^a, \mathbf{F}^b) = \frac{|\mathbf{F}^a \cup \mathbf{F}^b| - |\mathbf{F}^a \cap \mathbf{F}^b|}{|\mathbf{F}^a \cup \mathbf{F}^b|}. \tag{4.29}$$

Feature Integration (denoted as f_I). The feature integration of an ensemble is the percentage of features that are included in at least one of the base classifiers, which can be defined as follow:

$$f_I(\mathbf{F}) = \frac{|\bigcup_{k=1}^{K} \mathbf{F}^k|}{d}, \tag{4.30}$$

where the feature set $\mathbf{F} \in \mathbb{R}^d$ in the ensemble is aggregated by the feature sets $\mathbf{F}^k$ $(k = 1, 2, ..., K)$ from base classifiers.

In *SecENS*, we employ boosting [43] during the training phase. Boosting works by sequentially applying a base classifier to train the updated weighted samples and aggregating all the outputs generated from the individual classifiers into the final prediction. At each iteration, the misclassified samples are assigned higher weights, so that at the next iteration, the classifier will focus more on learning those samples [57]. With the use of boosting, our proposed ensemble learning approach *SecENS* builds the ensemble by integrating both feature differentiation (f_D) and feature integration (f_I) to diversify the classifiers while preserve a significant integration of whole feature space. Algorithm 7 illustrates the implementation of the proposed *SecENS* in detail.

4.4.4 Experimental Results and Analysis

In this section, to empirically validate our developed system *SecureDroid*, we present four sets of experimental studies using real sample collections obtained from Comodo Cloud Security Center: (1) In the first set of experiments, we evaluate the effectiveness of *SecureDroid* against different kinds of adversarial attacks; (2) In the second set of experiments, we assess the security of our proposed feature selection method *SecCLS* applied in the system *SecureDroid*; (3) In the third set of experiments, we further compare *SecureDroid* with other alternative defense methods; (4) In the last set of experiments, we evaluate the scalability of *SecureDroid* based on a larger sample collection.

Experimental Setup

Data collection The real sample collections we obtained from Comodo Cloud Security Center contain two sets: (1) The first sample set includes 8,046 apps (4,729 are benign apps, while the remaining 3,317 apps are malware including the families of Geinimi, GinMaster, DriodKungfu, Hongtoutou, FakePlayer, etc.). The extracted features from this sample set is with 926 dimensions, which include 104 permissions, 204 filtered intents, 330 API calls, and 288 new-instances. (2) The second dataset has larger sample collection containing 72,891 Android apps (40,448 benign apps and 32,443 malicious apps).

Evaluation Measures To quantitatively validate the effectiveness of different methods in Android malware detection, we use the performance indices shown in Table 4.4.

Table 4.4: Performance indices of Android malware detection

Indices	Description
TP	Number of apps correctly classified as malicious
TN	Number of apps correctly classified as benign
FP	Number of apps mistakenly classified as malicious
FN	Number of apps mistakenly classified as benign
Precision	$TP/(TP + FP)$
Recall/TPR	$TP/(TP + FN)$
ACC	$(TP + TN)/(TP + TN + FP + FN)$
F1	$2 \times Precision \times Recall/(Precision + Recall)$

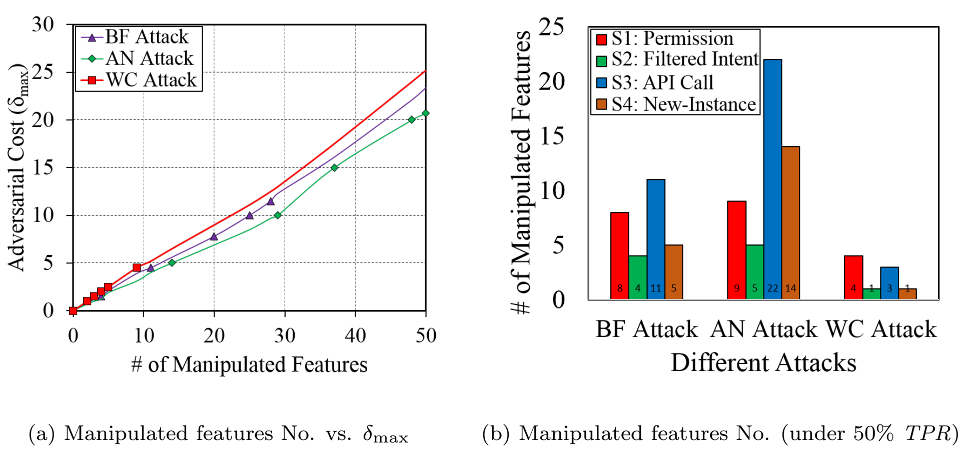
Implementation of different adversarial attacks To thoroughly assess the security and detection accuracy of our developed system *SecureDroid* against a wide class of attacks, we define and implement three kinds of representative adversarial attacks [87, 152, 83] considering different skills and capabilities of attackers, which are presented as followings.

Brute-force (BF) attack. To implement such kind of attack, for each malicious app (i.e., $\mathbf{x}^+$) we would like to manipulate, we first use Jaccard similarity [59] to find its most similar benign app (i.e., $\mathbf{x}^-$) from the sample set. Given these two apps, the procedure begins with $\mathbf{x}^+$ and modifies features one at a time to match those of $\mathbf{x}^-$, until the malicious app is classified as benign or the adversarial cost reaches to $\delta_{\max}$.

Anonymous (AN) attack. To simulate anonymous attack in which the defenders may have zero knowledge of what the attack is, we randomly manipulate some features for

addition and some for elimination with the adversarial cost of $\delta_{\max}$.

Well-crafted (WC) attack. In this adversarial setting, we use the wrapper-based approach [152, 83] to iteratively select a feature and greedily update this feature to incrementally increase the classification errors of the targeted learning system. Specifically, we first rank the features using methods such as *information gain* [59] to calculate their contributions to the classification problem. Then we conduct bi-directional feature selection, i.e., forward feature addition and backward feature elimination, to manipulate the malicious apps. At each iteration, using the attack model formulated in Eq. 4.5 (in Section 4.2.3) which encodes two competing objectives (i.e., maximizing the classification error while minimizing the adversarial cost for optimal attacks), a feature will be either added or eliminated.



(a) Manipulated features No. vs. $\delta_{\max}$

(b) Manipulated features No. (under 50% *TPR*)

Figure 4.11: Effectiveness evaluation of different attacks.

To estimate the effectiveness of different attacks, we implement the above three kinds of attacks to access the *Original-Classifier* described in Section 4.1 and make its *TPR*s drop from 90% to 50%. For each attack, Figure 4.11(a) shows the relations between the numbers of manipulated features and the corresponding adversarial costs which also consider the complexity of different feature manipulations. Among these attacks, the WC attack is the most effective strategy, since the adversarial cost of this attack (also the number of features manipulated by this attack) is minimum when compromising the learning classifier into the same level, which can be seen in Figure 4.11(b).

Evaluation of *SecureDroid* against Different Adversarial Attacks

In this set of experiments, based on the first sample set described in Section 4.4.4, we validate the effectiveness of *SecureDroid* against above mentioned adversarial attacks.

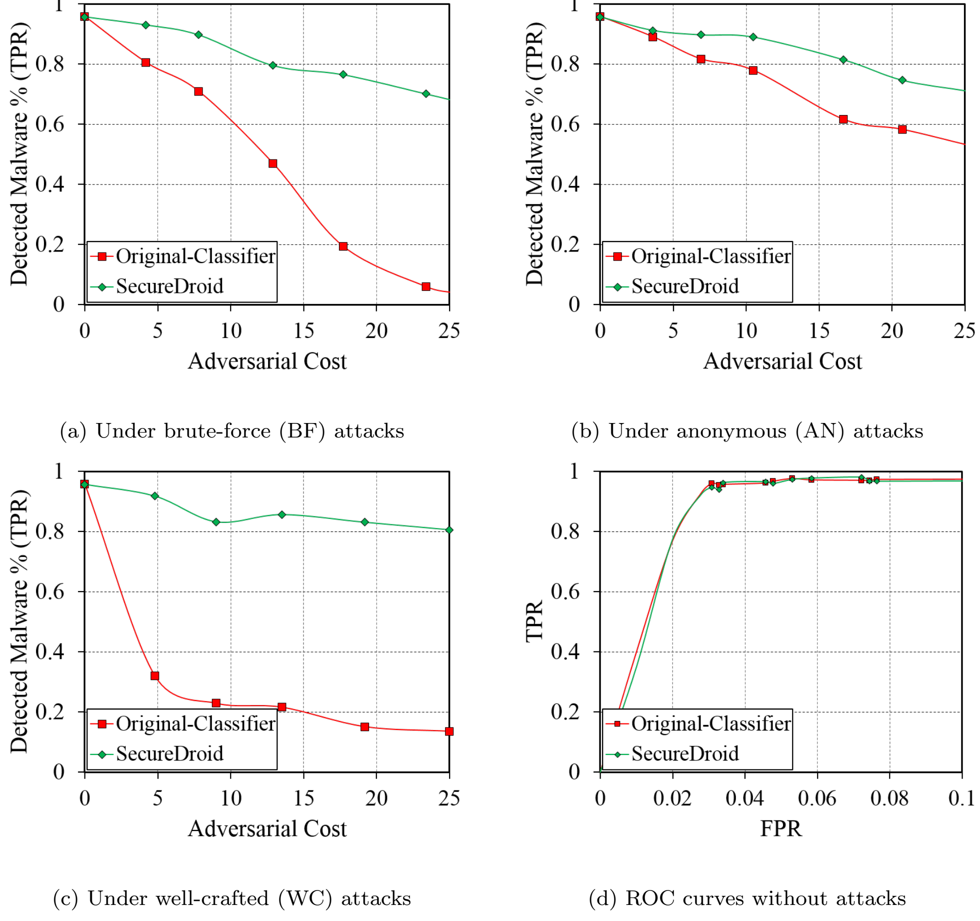
To estimate the reasonable adversarial cost for attackers to perform the adversarial attacks, based on the first sample set, we explore the average number of features that each app possesses, which is 98. In general, 50% of the average number of features is considered as an extreme for the adversary to perform the attack. Based on these observations, we implement the above three kinds of attacks to access both *SecureDroid* and *Original-Classifier* with the manipulated features varying in $\{10\%, 20\%, 30\%, 40\%, 50\%\}$ of the average number of features (i.e., 98), whose corresponding adversarial costs under different kinds of attacks are shown in Figure 4.12.(a)–(c) (X-axis). We randomly select 90% of the samples for training, while the remaining 10% is used for testing. We use these attacks to taint the malicious apps in the testing set respectively, and then assess the security of *SecureDroid* under different attacks with different adversarial costs by comparison with the *Original-Classifier*. To implement *SecureDroid*, empirically we found that the parameters of $\lambda = 0.7$ and $\rho = 0.8$ in Eq. 4.21 are the best, and apply them to our problem throughout the experiments. To validate the detection performance of *SecureDroid* without attacks, we also perform 10-fold cross validations for evaluation. The experimental results are shown in Figure 4.12.

Under attacks. From Figure 4.12(a)–(c), we can see that *SecureDroid* can significantly enhance security compared to the *Original-Classifier*, as its performance decreases more elegantly against increasing adversarial costs, especially in the scenarios of BF attack and WC attack. In the BF attack, the *TPR* of *Original-Classifier* drops to 5.99% with adversarial cost $\delta_{\max}$ of 23.4 (i.e., modifying 50 features), while *SecureDroid* retains the *TPR* at 70.06% with the same adversarial cost. In the WC attack, the performance of *Original-Classifier* is compromised to a great extent with *TPR* of 13.62% under the adversarial cost of 25.2; instead, *SecureDroid* can significantly bring the detection system back up to the desired performance level: the *TPR*s of *SecureDroid* are actually never lower than 80.00% even with increasing adversarial costs. This demonstrates that *SecureDroid* which integrates our proposed methods is resilient against the most effective attack strategy (i.e., WC attack) among the three representative adversarial attacks. In the AN attack, which is simulated under defenders have zero knowledge of what the attack is and by randomly injecting or removing features from the malicious apps, *SecureDroid* also outperforms the *Original-Classifier*, which can retain the average *TPR* at 85.16% with different adversarial costs.

Without attacks. Figure 4.12(d) shows the ROC curves of the 10-fold cross validations for *Original-Classifier* and *SecureDroid* without any attacks. From Figure 4.12(d),

we can see that, though *SecureDroid* is designed to be resilient against different kinds of adversarial attacks, its detection performance is as good as the *Original-Classifier* in the absence of attacks.

The experimental results and above analysis demonstrate that *SecureDroid* can effectively enhance security of the learning-based classifier without compromising the detection accuracy, even attackers may have different knowledge about the targeted learning system. Based on these properties, *SecureDroid* can be a resilient solution in Andriod malware detection.



(a) Under brute-force (BF) attacks

(b) Under anonymous (AN) attacks

(c) Under well-crafted (WC) attacks

(d) ROC curves without attacks

Figure 4.12: Security evaluations under brute-force (BF) attacks, anonymous (AN) attacks, well-crafted (WC) attacks, and without attacks.

Evaluation of *SecCLS* Applied in *SecureDroid*

In this section, based on the same training and testing datasets in the previous section, we further validate the effectiveness and significance of our proposed feature

selection method *SecCLS* in building a more secure classifier. We compare *SecureDroid* which applies *SecCLS* to select features for each base classifier with ensemble of random feature selection (denoted as *ERFS*) that uses random feature selection method to construct base classifiers [16, 63], in the settings of under attacks and without attacks. As illustrated in Section 4.4.4, since well-crafted (WC) attack is the most effective attack strategy among those three, we evaluate the *SecureDroid* and *ERFS* under such kind of attacks. The experimental results are shown in Table 4.5.

Table 4.5: Comparison of *SecureDroid* with *SecCLS* and *ERFS* with random feature selection against well-crafted attacks (UnderAtt) and without attacks (NonAtt).

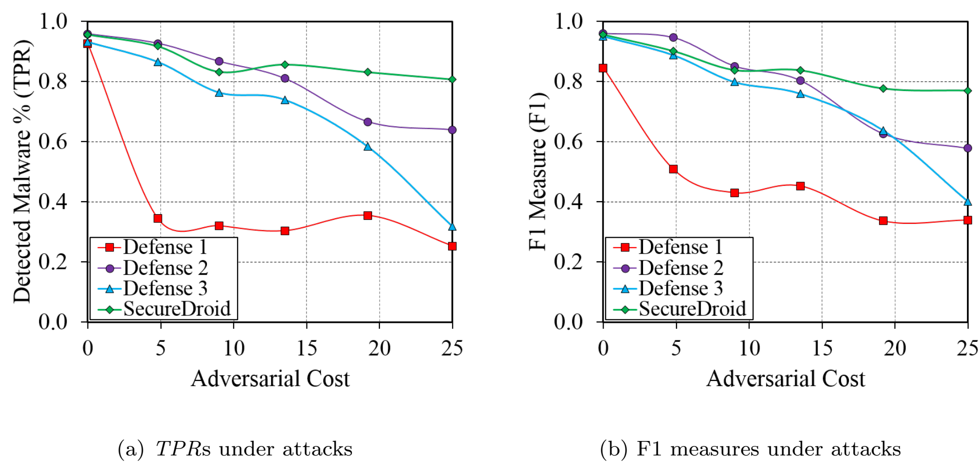
		NonAtt	UnderAtt [$\delta_{\max}$ (features modified)]				
			4.8(10)	9.0(20)	13.5(30)	19.2(40)	25.2(50)
ERFS	TPR	0.9072	0.8563	0.5045	0.4326	0.2934	0.1647
	ACC	0.9230	0.9354	0.7888	0.7559	0.6981	0.6509
	F1	0.9072	0.9167	0.6647	0.5953	0.4465	0.2813
SecureDroid	TPR	0.9566	0.9177	0.8323	0.8563	0.8308	0.8069
	ACC	0.9634	0.9168	0.8665	0.8621	0.8019	0.8106
	F1	0.9559	0.9015	0.8380	0.8375	0.7768	0.7795

From Table 4.5, we can observe that, (1) **Under attacks:** *ERFS* can somehow be resilient against the attack (with *TPR* of 85.63%) when the adversarial cost is small ($\delta_{\max} = 4.8$, modifying 10 features). However, with the increasing adversarial costs, the detection performance of *ERFS* drops drastically (e.g., its *TPR* drops to 16.47% when the adversarial cost $\delta_{\max}$ is 25.2 corresponding to manipulating 50 features). In contrast, *SecureDroid* using *SecCLS* for feature selection can significantly enhance security, as its performance decreases more elegantly against increasing adversarial costs and its *TPR*s are actually never lower than 80.00% with different adversarial costs. The reason behind this is that *SecCLS* integrated in *SecureDroid* reduces the possibility of selecting those features attackers tend to manipulate, i.e., to achieve same attack utility, *SecCLS* will force attackers to modify larger number of features compared with random feature selection method. (2) **Without attacks:** *SecureDroid* also performs better than *ERFS* in the absence of attacks (i.e., about 4-5% higher detection accuracy). This is because, compared with *ERFS* which randomly assigns equal probability for each feature being selected, *SecureDroid* applying *SecCLS* is capable to retain majority of the features for

each individual classifier and thus assure its detection accuracy without attacks.

Comparisons of *SecureDroid* with Other Defense Methods

In this set of experiments, we further examine the effectiveness of *SecureDroid* against the adversarial attacks (i.e., well-crafted attack as it shows most effective) by comparisons with other popular defense methods, including (1) ***feature evenness*** (denoted as ***Defense1***) which enables the *Original-Classifier* to learn more evenly-distributed feature weights using the method proposed in [78]; (2) ***classifier retraining*** (denoted as ***Defense2***) which follows Stackelberg game theories [19, 58, 18, 127] and models the attack as a vector $\boldsymbol{\theta}$ to modify the training data set $\mathbf{X}$ where the *Original-Classifier* is retrained [127, 133]; (3) ***classifier built on reduced feature set*** (denoted as ***Defense3***) which carefully selects a subset of features based on the generalization capability of the *Original-Classifier* and its security against data manipulation applying the method proposed in [152]. The experimental results are reported in Figure 4.13.



(a) *TPR*s under attacks

(b) F1 measures under attacks

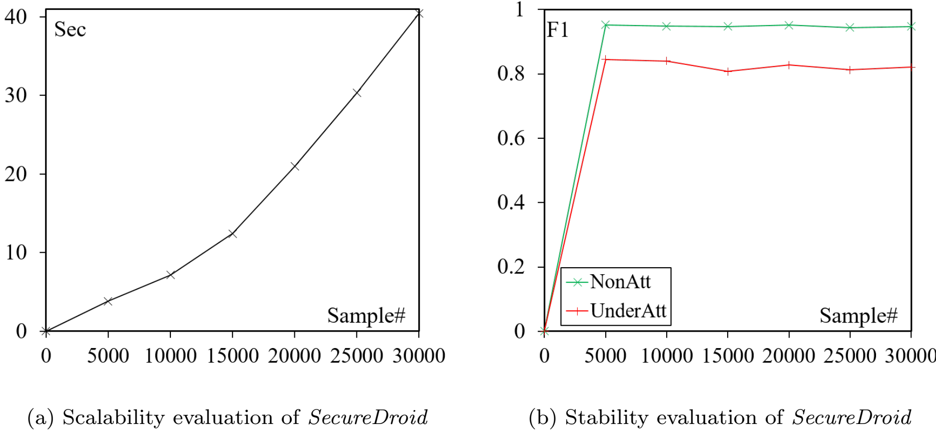
Figure 4.13: Comparisons of different defense methods.

From Figure 4.13, we can see that *SecureDroid* significantly outperforms the other defense models (i.e., *Defense1–3*) against the well-crafted attacks. Although ***Defense2*** (i.e., classifier retraining) performs slightly better than *SecureDroid* when the adversarial costs $\delta_{\max} \in \{4.8, 9.0\}$ (i.e., modifying 10 and 20 features), the difference is not statistically significant. In fact, the retrained model modifies the training data distribution approximate to the testing space through the attack model $\boldsymbol{\theta}$. After modifying a large number of features in the malicious apps, the model tends to produce a distribution that is very close to that of the benign apps. In this case, the retrained model may

not be able to differentiate benign and malicious apps accurately. From Figure 4.13, we also observe that as the adversarial cost $\delta_{\max}$ increases, the performance of the retrained model suffers a great drop-off. For *Defense1* and *Defense3*, their performances (*TPR*s and *F1* measures in Figure 4.13) sharply degrade when adversarial cost increases. For ***Defense1***, the weight evenness merely exploits the information of the classifier's feature weights while ignoring manipulation costs of different features; for ***Defense3***, the model is built on a carefully selected feature subset, whose robustness could be compromised when attackers manipulate a certain number of these features.

Scalability evaluation of *SecureDroid*

In this section, based on the second sample set with larger size described in Section 4.4.4 which consists of 72,891 apps (32,443 malicious and 40,448 benign), we systematically evaluate the performance of our developed system *SecureDroid*, including scalability and detection effectiveness. We first evaluate the training time of *SecureDroid* with different sizes of the training sample sets. Figure 4.14(a) presents the scalability of our developed system. We can observe that as the size of the training data set increases, the running time for our detection system is quadratic to the number of training samples. When dealing with more data, approximation or parallel algorithms could be developed. Figure 4.14(b) shows the detection stability of *SecureDroid* against the adversarial attacks (i.e., well-crafted attacks) and in the absence of attacks, with different sizes of sample sets. From the results, we can conclude that our developed system *SecureDroid* can enhance security of machine learning based detection, and is feasible in practical use for Android malware detection against adversarial attacks.



(a) Scalability evaluation of *SecureDroid* (b) Stability evaluation of *SecureDroid*

Figure 4.14: Scalability and stability evaluation of *SecureDroid*.

4.5 *DroidEye*: Fortifying Learning Security over Feature Space

SecureDroid has provided a significant solution to enhance the security of machine learning-based classifier against adversarial attacks, which is independent from the skills and capabilities of the attacks to some extent. The limitations of *SecureDroid* lie in that: (1) adjustable parameters, and manipulation costs are empirically decided based on the training data; (2) feature manipulation methods (addition or elimination) are determined through the assumption that attackers conduct a well-crafted attack and are able to utilize information gain or max-relevance to calculate different contributions of the features for the classification of malicious and benign files respectively.

In this section, we want to weaken the assumption of feature manipulations (i.e., adjustable parameters, manipulation costs, and feature manipulation methods) and construct a more resilient and flexible solution against the advanced attacks. Resting on a set of features (i.e., permissions, filtered intents, application attributes, API calls, new-instances, and exceptions) extracted from the Android apps, to harden the evasion, we first present *count featurization* [81, 117] to transform the binary feature space into continuous probabilities that encode the data distribution; to improve the system security while not compromising the detection accuracy, we further introduce *softmax function with adversarial parameter* for model construction. Accordingly, we develop a system called *DroidEye* which integrates the proposed method to fortify security of learning-based classifier against adversarial malware attacks. The system architecture of *DroidEye* is shown in Figure 4.15.

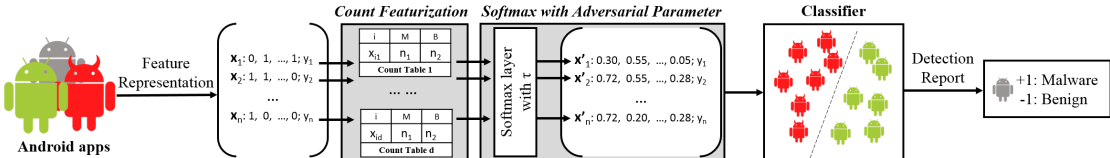


Figure 4.15: An overview of system architecture of *DroidEye*. In the system, the collected apps are first represented as d-dimensional binary feature vectors. To harden the evasion, *count featurization* is used to transform each binary feature vector $\mathbf{x}_i$ to a continuous feature vector $\mathbf{x}'_i$; then *softmax function with adversarial parameter* is introduced to find the best trade-off between security and accuracy for the classifier. For a new app, after feature representation, it will be predicted as either benign or malicious using the classifier.

4.5.1 Feature Representation

In this section, we will still use Android apps as a case study to investigate the secure-learning paradigm *DroidEye*. As introduced in Section 4.4.1, permissions, filtered intents, API calls, and new-Instances can reflect the behaviors and the interaction between an app and other apps or operation system, and thus have been extracted to represent apps. In the manifest file and dex file, there are some more useful information that can be gleaned: in manifest file, the components are first configured using a set of *application attributes* to set default values for corresponding elements (e.g., whether allow the app to reset user data); the dex file also utilizes *exceptions* to indicate conditions that an app may want to catch. In this respect, we further extract *application attributes* and *exceptions* as features. Accordingly, we have six sets of features (***S1–S6*** shown in Table 4.6) to represent Android apps that include: permissions (*S1*), filtered intents (*S2*), and application attributes (*S3*) from manifest files, API calls (*S4*), new-instances (*S5*), and exceptions (*S6*) from dex files.

Table 4.6: Illustration of extracted features for Android apps in *DroidEye*

	Features	Examples
Manifest	*S1:* Permissions	*INTERNET*
	S2: Filtered Intents	*action.MAIN*
	S3: Application Attributes	*debuggable*
Dex	*S4:* API calls	*containsHeader*
	S5: New-Instances	*util/HashMap*
	S6: Exceptions	*SecurityException*

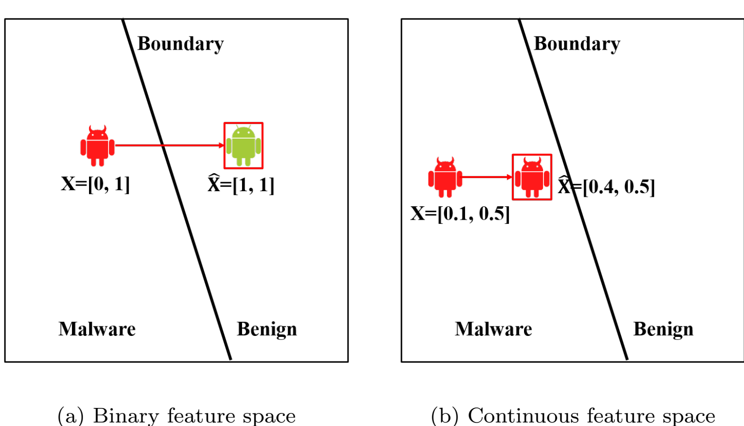
Let d be the number of all extracted features in *S1* – *S6* in dataset D. Each app can then be represented by a binary feature vector:

$$\mathbf{x}_i = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \rightarrow \begin{matrix} \text{READ_PHONE_STATE} \\ \text{vending.INSALL_REFERER} \\ \text{allowClearUserData} \\ \text{getSimSerialNumber} \\ \text{Landroid/app/ProgressDialog} \\ \text{ArithmeticException} \end{matrix} \begin{matrix} \}\ S1 \\ \}\ S2 \\ \}\ S3 \\ \}\ S4 \\ \}\ S5 \\ \}\ S6 \end{matrix}$$

where $\mathbf{x}_i \in \mathbb{R}^d$, and $x_{ij} = \{0, 1\}$ (i.e., if app i includes feature j, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$).

4.5.2 Count Featurization

By performing AdvAttack described in Section 4.3.2, attackers may autonomously add a feature in the app (i.e., set 0 to 1 in the vector). For example, they can add permissions in the manifest file without influence on other existing functionalities; they can also inject API calls in the methods which will be never called by any invoke instructions in the dex file. Figure 4.16(a) shows an example that attackers can successfully generate a variant ($\hat{\mathbf{x}} = [1, 1]$) to evade the detection by injecting a feature in the original malicious app (denoted as $\mathbf{x} = [0, 1]$). But from the defenders' point of view, if the binary feature space is featurized into continuous space of each feature value being $0 \leq x \leq 1$, the actual gradient of the feature addition or elimination available to the attackers may be significantly squashed. If adversarial gradients are low, crafting adversarial attacks becomes more difficult because small feature manipulations will not induce high output variations for the learning model [100], which thus makes the model more resilient against the adversarial attacks. As shown in Figure 4.16(b), with the same manipulation from $\mathbf{x}$ to $\hat{\mathbf{x}}$, the step towards the boundary is sufficiently shortened in the continuous feature space, which makes the evasion fail. This intuition of *gradient masking* [99] inspires us to design a secure defense with count featurization [81] to combat the adversarial attacks.



(a) Binary feature space (b) Continuous feature space

Figure 4.16: Defenses in different feature spaces.

Count Featurization Count featurization is originally motivated by the objective of reducing training time on data that contains categorical features by feeding learning

algorithms with a limited subset of the collected data combined with historical collections from much larger amounts of data [81, 117]. The general idea of this technique is to featurize the data with the conditional probability of the class given the frequency (i.e., the number of times) a feature value was observed with each class, instead of directly using the value of a categorical feature [81]. Given a binary feature vector of an app $\mathbf{x}$, to perform count featurization, count tables for each feature are first aggregated on the original dataset. The conditional probabilities are then calculated directly from the count tables as defined below.

Definition 4.1 **Count table** is designed per feature in $\mathbf{x}$. It maintains the number of malicious apps for each feature value (denoted as $M(x_i)$) and the number of benign apps for each feature value (denoted as $B(x_i)$); it therefore encodes each feature's propensity to malware and benign apps.

Definition 4.2 To count-featurize a binary feature vector $\mathbf{x} = \langle x_1, x_2, ..., x_d \rangle$, **count featurization** projects each of its features with the conditional probabilities calculated from the count tables, i.e., $\mathbf{x}' = \langle \mathcal{P}(M(x_1)|x_1), ..., \mathcal{P}(M(x_d)|x_d) \rangle$, where $\mathcal{P}(M(x_i)|x_i) = M(x_i)/(M(x_i) + B(x_i))$ from the row matching x_i in the corresponding count table.

This is a simplified version of the count featurization function, which is particularly valuable when the features are of high cardinality [81]. Considering that each feature only has two values (i.e., 1 and 0) in our application, this potentially is at the cost of reducing predictive accuracy. To preserve each feature's informative property, we formulate a softmax function to convert conditional probabilities into more effective action probabilities for model construction. The softmax function for feature x_i is given by

$$\bar{\mathcal{P}}(x_i) = \frac{\exp(\mathcal{P}(M(x_i)|x_i)/\tau)}{\sum_{k \in \{M(x_i), B(x_i)\}} \exp(\mathcal{P}(k|x_i)/\tau)}, \tag{4.31}$$

where τ is an adjustment parameter that plays a critical role to actively keep the trade-off between security and accuracy for the classifier trained on the count-featurized probability vectors. In adversarial settings, we refer to this adjustment parameter as the *adversarial parameter*. The higher the adversarial parameter of softmax function is, the more ambiguous and secure its action probabilities will be (i.e., when $\tau \to +\infty$, all the probabilities are close to 0.5), whereas the smaller τ is, the more discrete and informative its probabilities will be (i.e., when $\tau \to 0^+$, the probabilities are close to 1 or 0) [100]. Therefore, based on the softmax function with adversarial parameter in Eq. (4.31), the

final probability vector for $\mathbf{x}$ can be formulated as

$$\mathbf{x}' = \langle \bar{\mathcal{P}}(x_1), \bar{\mathcal{P}}(x_2), ..., \bar{\mathcal{P}}(x_d) \rangle. \tag{4.32}$$

Figure 4.17 shows an example of an app $\mathbf{x}$ and its count-featurized vector $\mathbf{x}'$ with $\tau = 0.5$.

Features: <INTERNET, …, debuggable>

INTERNET	M(x)	B(x)
1	100	1,000
0	300	200

debuggable	M(x)	B(x)
1	20	400
0	380	800

X = <1, …, 0> → X′ = <0.163, …, 0.329>

$\bar{P}$(INTERNET=1)=exp((100/1,100)/0.5) / [exp((100/1,100)/0.5) + exp((1,000/1,100)/0.5)]

$\bar{P}$(debuggable=0)=exp((380/1,180)/0.5) / [exp((380/1,180)/0.5) + exp((800/1,180)/0.5)]

Figure 4.17: An example of count featurization.

Proposed Defense An adversary-aware learning system for Android malware detection should (1) relatively consistently predict the correct labels for the manipulated apps, as well as (2) significantly display good accuracy on benign apps [100, 134]. Different from the previous work towards this goal which substantially performed model regularization [32], data retraining [54, 133, 83], or feature reduction [152, 82], we adapt count featurization to improve the security of the learning model while leaving model, training data, and feature sets unchanged. It's recalled that the benefit of count featurization in our application is intuitive as the probabilities ranging in $[0, 1]$ encode additional distribution information about each class, in addition to simply providing an app's feature existences, permitting more secure and accurate learning. To implement the defense, called *DroidEye*, we add a count featurization layer and a softmax layer with adversarial parameter τ in front of the learning model shown in Figure 4.9, which count-featurizes the binary feature vector $\mathbf{x}$ for each app into continuous vector $\mathbf{x}'$. The learning model predicts the class for a given app by training on count-featurized conditional probabilities. Note that, when classifying a new app, the adversarial parameter τ should be configured as a low value (e.g., $\tau = 1$) to make the predictions more accurate.

Algorithm 8 illustrates the implementation of the proposed defense (denoted as *DroidEye*) in detail. Since *DroidEye* has not changed the original model and training data, the only impact on computational complexity is limited for count featurization, requiring $O(nd)$ queries, which ensures that the learning model can still take advantage of large dataset to achieve the good performance.

Algorithm 8: *DroidEye* - A secure classifier with count featurization against adversarial Android malware attacks.

Input: Training data set $D = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$; τ: adversarial parameter

Output: $\mathbf{f}$: the labels for the apps

Formulate count tables for features: $M(\mathbf{X})$ and $B(\mathbf{X})$;

Initialize($i = 1$);

for $i \leq n$ **do**

  Calculate $\langle \mathcal{P}(M(x_1)|x_1), ..., \mathcal{P}(M(x_d)|x_d) \rangle$;

  Calculate $\mathbf{x}'_i = \langle \bar{\mathcal{P}}(x_1), \bar{\mathcal{P}}(x_2), ..., \bar{\mathcal{P}}(x_d) \rangle$;

  $\mathbf{x}_i = \mathbf{x}'_i$;

end

Use conjugate gradient descent method to solve:

$\underset{\mathbf{X},\mathbf{w},\mathbf{b}}{\text{argmin}}\ \mathcal{L}(\mathbf{y}, f(\mathbf{X})) + \beta||\mathbf{w}|| + \gamma||\mathbf{b}||$;

return $\mathbf{f} = \text{sign}(f(\mathbf{X}))$;

Theoretical Analysis The adversary generally takes two steps to craft the adversarial attack: (1) evaluate the sensitivity of class change to each input feature, and (2) use the sensitivity information to select a set of manipulations among the input features [100]. In the attacks (e.g., FGSM) discussed in Section 4.5.3, the sensitivity of the model to the feature manipulations is primarily evaluated through adversarial gradient, which is defined as the gradient difference between the adversarial attack and the original malware:

$$\nabla G = \nabla \mathcal{L}(f(\hat{\mathbf{x}}), y) - \nabla \mathcal{L}(f(\mathbf{x}), y). \tag{4.33}$$

The higher adversarial gradient denotes that crafting adversarial attacks is relatively easier as small feature manipulations will induce high output variation of the learning model [100]. The adversarial gradient will not vanish unless $\nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}), y)$ becomes zero, which is impractical [96]. But count featurization can significantly reduce ∇G to the small feature manipulations.

Again, our defense using continuous probability vectors by count featurization benefits from the additional knowledge found in the apps. The additional knowledge encodes the relative distributions of malware and benign apps, which prevents the models from fitting too tightly to the feature existences, and contributes to a more stable while still accurate feature representations around training data. On the contrary, the adversary

may only manage to add or eliminate a small number of features to craft the attacks $\hat{\mathbf{x}}$, which may have limited impact on the actual probability distributions and data structure, that is, based on the same feature manipulations, roughly

$$\nabla\mathcal{L}(f(\hat{\mathbf{x}}'), y) - \nabla\mathcal{L}(f(\mathbf{x}'), y) < \nabla\mathcal{L}(f(\hat{\mathbf{x}}), y) - \nabla\mathcal{L}(f(\mathbf{x}), y). \tag{4.34}$$

Actually when the probabilities $\mathbf{x}'$ are all smoothed to be close to 0.5, $\nabla G_{\mathbf{x}'}$ would be significantly approaching 0. If the small feature manipulations cannot induce the evasion, the adversary may have to manipulate a larger number of features to achieve the goal. Considering the adversarial cost, and the app's original functionalities, this may not be always feasible.

Note that the count featurization is controlled by an adversarial parameter τ in softmax, which is capable of further adjusting the trade-off between the smoothness and accuracy of the learning model. Here, we further quantify the continuous feature space's smoothness to the input $\mathbf{x}$ by its Jacobian Matrix [100]. We use $\bar{\mathcal{P}}_i(x)$ to denote the probability of feature x to be with class i ($i \in \{malware, benign\}$), and let $G(x) = \exp(M(x)/\tau) + \exp(B(x)/\tau)$. Its formulation of component (i, j) at adversarial parameter τ is:

$$\begin{aligned}
\left.\frac{\partial\bar{\mathcal{P}}_i(x)}{\partial x_j}\right|_{\tau} &= \frac{\partial}{\partial x_j}\left(\frac{\exp(M(x)/\tau)}{\exp(M(x)/\tau) + \exp(B(x)/\tau)}\right) \\
&= \frac{1}{G^2(x)}\left(\frac{\partial \exp(M(x)/\tau)}{\partial x_j}G(x) - \exp(M(x)/\tau)\frac{\partial G(x)}{\partial x_j}\right) \\
&= \frac{\exp(M(x)/\tau)\exp(B(x)/\tau)}{\tau G^2(x)}\left(\frac{\partial(\exp(M(x) - \exp(B(x))}{\partial x_j}\right)
\end{aligned} \tag{4.35}$$

Since $M(x)$ and $B(x)$ are fixed values for each feature, and the component values are inversely proportional to τ, the increasing τ will essentially reduce the values of all the components of Jacobian matrix. This analysis illustrates that count featurization resting on high settings of τ reduces the model sensitivity to small feature manipulations. When τ is well tuned, the model may also preserve the reasonable generalization ability. The empirical analysis will be given in Section 4.5.3.

4.5.3 Experimental Results and Analysis

In this section, we present three sets of experimental studies to empirically validate our developed system *DroidEye*. The real sample collection we obtained from Comodo Cloud Security Center contains 14,804 apps (8,059 are benign apps, while the remaining 6,745 apps are malware including the families of Geinimi, GinMaster, DriodKungfu, etc.)

with 812 features, including 105 permissions, 68 filtered intents, 8 application attributes, 330 API calls, 259 new-instances, and 42 exceptions. We randomly select 90% of the samples for training, while the remaining 10% is used for testing. To quantitatively validate the effectiveness of different methods in Android malware detection, we use the performance indices shown in Table 4.4.

To thoroughly assess the security and detection accuracy of *DroidEye* against a wide class of attacks, we implement three kinds of representative adversarial attacks:

- *AdvAttack*, introduced in Section 4.3.2, is L_0 attack model [21] that measures $\mathcal{C}(\hat{\mathbf{x}}, \mathbf{x})$ using L_0 distance.
- *Fast Gradient Sign Method (FGSM)* [54] is one of L_∞ attacks [21] that measure $\mathcal{C}(\hat{\mathbf{x}}, \mathbf{x})$ using L_∞ distance. As an L_∞ attack model, given a malicious app $\mathbf{x}$, Fast Gradient Sign Method (FGSM) [54] sets

$$\hat{\mathbf{x}} = \mathbf{x} + \varepsilon \cdot \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}), y)), \tag{4.36}$$

  where $\mathcal{L}$ is the loss function used in classifier training, y is the target label for $\mathbf{x}$, and ε is a constant parameter. Intuitively, for each feature x_i, FGSM uses the gradient of the loss function to determine in which direction the feature's value should be increased or decreased to minimize the loss function. To apply FGSM to the binary feature space in our application, we further define a threshold θ to adjust $\hat{\mathbf{x}}$ so that $\hat{x}_i = \{0, 1\}$, i.e., if $\hat{x}_i \geq \theta$, then $\hat{x}_i = 1$; otherwise $\hat{x}_i = 0$. In this paper, we implement FGSM to conduct L_∞ attack for our further investigation.
- *ANAttack*, an anonymous attack, randomly manipulates some features for addition and elimination to simulate the attack in which the defenders may have zero knowledge of what the attack is.

Evaluation of *DroidEye* with Different Adversarial Parameter Values

In this set of experiments, we evaluate how different settings of the adversarial parameter τ in the count featurization function may influence the performance of our developed system *DroidEye*. Note that the adversarial parameter is set to 1 when count-featurizing the testing apps. That is, τ only impacts on model training. It's recalled that the adversarial parameter is the key to adjust the trade-off between the smoothness and accuracy of the learning model. Therefore, the objective here is to identify the optimal training adversarial parameter for *DroidEye* resting on our data collection. Here, we

specifically explore AdvAttack with different numbers of manipulated features to taint the malicious apps in the testing set, and repeat the experiments by measuring the adversarial parameter τ varying in $\{0.1, 0.5, 1, 1.5, 2, 5, 6, 7, 10\}$. The experimental results are shown in Figure 4.18.

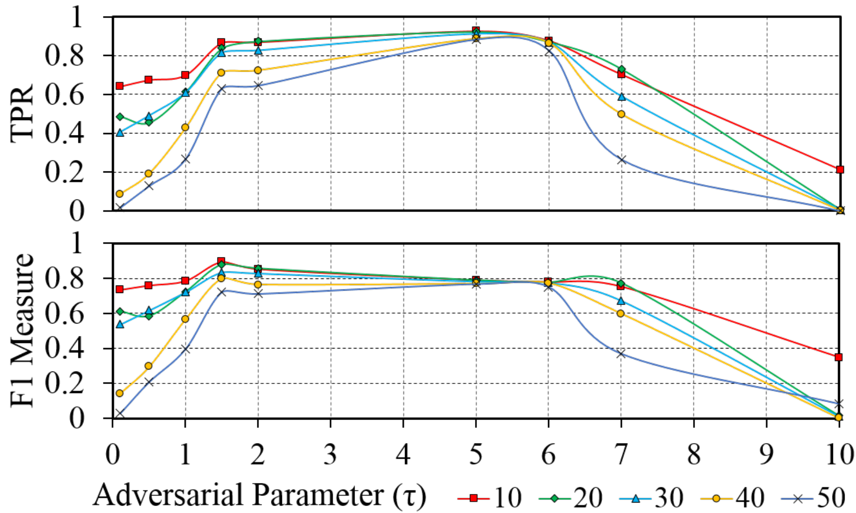


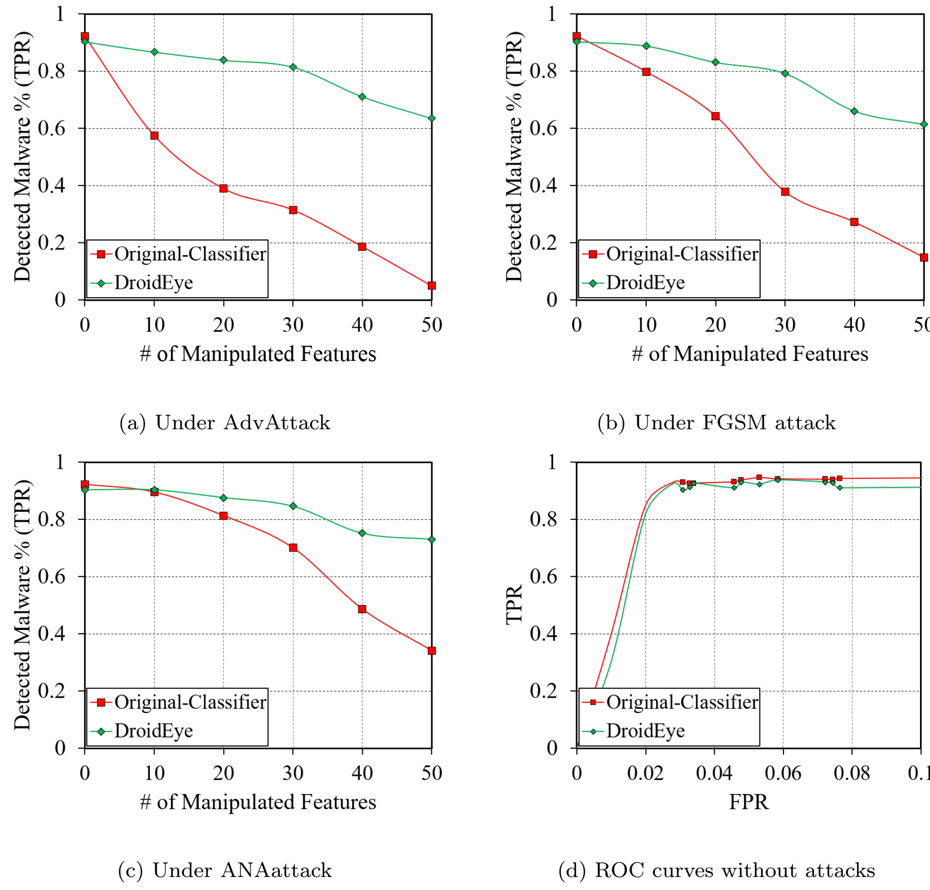
Figure 4.18: Evaluation of *DroidEye* with different τ under AdvAttack with number of manipulated features varying from 10 to 50.

From Figure 4.18, we can see that: (1) when $\tau \rightarrow 0^+$, the learning model is fairly vulnerable to the adversarial attacks, since the probability values in the feature space are extremely close to 1 or 0; increasing the parameter generally increases the *TPR*s while making adversarial evasion harder; (2) there is a turning point after the *TPR*s reach the highest (around $\tau = 5.5$); as $\tau \rightarrow 10$, the *TPR*s suffer from a drastic drop for all the probability values are approaching 0.5, which makes the features too ambiguous to discriminate malware from the benigh apps. Observations validate our theoretical analysis in Section 4.5.2. To fortify the security of the learning model while not compromising the detection accuracy, the optimal adversarial parameter should be linked to both *precision* and *recall*. In Figure 4.18, we can observe *F1* measures at $\tau = 1.5$ outperform the others with the highest average value (i.e., an average of 0.8254). Hence in the following experiments, we will formalize *DroidEye* based on the setting of $\tau = 1.5$.

Evaluation of *DroidEye* against Different Attacks

In this section, we validate the effectiveness of *DroidEye* against above mentioned adversarial attacks. We learn a linear SVM (denoted as *Original-Classifier*) as the learning-

based classifier to facilitate our empirical analysis. To estimate the impact of feature manipulations on both *DroidEye* and *Original-Classifier*, we implement the above three kinds of attacks with the number of manipulated features varying in $\{10, 20, 30, 40, 50\}$, and then assess the security of *DroidEye* under different attacks by comparisons with *Original-Classifier*. To validate the detection accuracy of *DroidEye* without attacks, we also perform 10-fold cross validations for evaluation. The experimental results are shown in Figure 4.19.



(a) Under AdvAttack (b) Under FGSM attack

(c) Under ANAattack (d) ROC curves without attacks

Figure 4.19: Security evaluations of *DroidEye* and *Original-Classifier* under AdvAattack, FGSM attack, ANAattack, and without attacks.

Security. Figure 4.19(a)–(c) signify that *DroidEye* can significantly enhance security compared to the *Original-Classifier*, as its performance decreases more elegantly against increasing manipulated features, especially in the scenarios of FGSM attack and AdvAttack. In the FGSM attack, the *TPR* of *Original-Classifier* drops to 14.94% with 50 manipulated features, while *DroidEye* retains the *TPR* at 61.37% with the same feature manipulations. In the AdvAttack, the performance of *Original-Classifier* is compromised

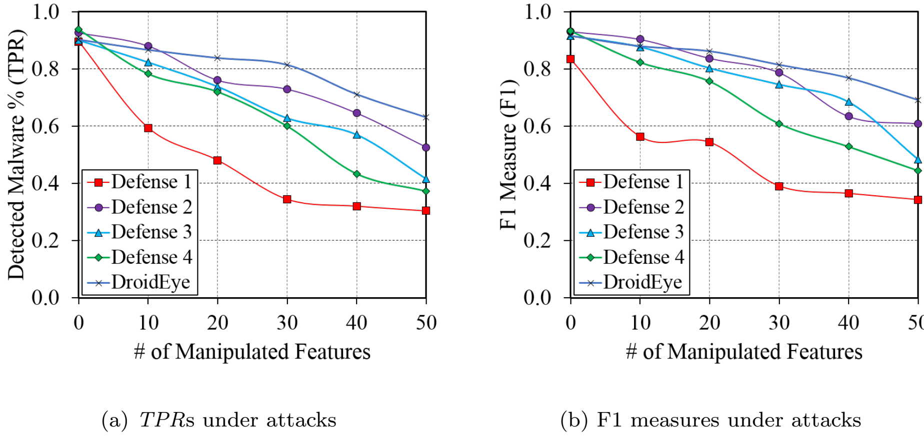
to a greater extent with *TPR* of 4.98% with 50 features manipulated; instead, *DroidEye* can significantly bring the detection system back up to the desired performance level: the average *TPR* of *DroidEye* are actually stay around 77.00%. This demonstrates that *DroidEye* which devises our proposed count featurization is indeed resilient against those representative attack strategies. In the ANAttack, which is simulated under defenders have zero knowledge of what the attack is and by randomly injecting or removing features from the malicious apps, *DroidEye* also outperforms the *Original-Classifier*, which can retain the average *TPR* at 82.12% with different feature manipulations.

Accuracy. Figure 4.19(d) shows the ROC curves of the 10-fold cross validations for *Original-Classifier* and *DroidEye* without any attacks, from which we can see *DroidEye* is not only resilient against adversarial attacks, but its detection accuracy (an average 0.9210 *TPR* at 0.0525 *FPR*) is also as good as the *Original-Classifier* in the absence of attacks.

According to the analysis of security and accuracy, *DroidEye* can effectively fortify security of the learning-based classifier against different representative types of adversarial attacks (e.g., L_0 attack, and L_∞ attack) while not compromising the detection accuracy. Considering that *DroidEye* improves the security of the learning model only through feature space while leaving model, training data and feature sets unchanged, it can be a feasible solution for real-world Android malware detection.

Comparisons of ***DroidEye*** with Other Representative Defense Methods

In this set of experiments, similar to the experimental comparisons of *SecureDroid* with other defenses in Section 4.4.4, we further examine the effectiveness of *DroidEye* against the adversarial attacks by comparisons with other popular defense methods, including (1) *feature evenness* (*Defense1*) which enables the *Original-Classifier* to learn more evenly-distributed feature weights through feature reweighting [78]; (2) *classifier retraining* (*Defense2*) which retrains the *Original-Classifier* using the adversarial examples [83, 133]; (3) *adversarial feature selection* (*Defense3*) which selects a subset of features based on the generalization capability of *Original-Classifier* and its security against data manipulation [152]; (4) *distillation* (*Defense4*) which applies the soft labels for training through softmax function devised in distillation layer [100]. As illustrated in Section 4.5.3, AdvAttack is the most effective attack strategy among those three. Thus here we evaluate different defense methods under such kind of attacks. The experimental results are reported in Figure 4.20.



(a) *TPR*s under attacks (b) F1 measures under attacks

Figure 4.20: Comparisons of different defense methods.

From Figure 4.20, we can observe that *DroidEye* performs better than the other defense models (i.e., *Defense1–4*) against AdvAttack. As expected, *Defense1*, *Defense2*, and *Defense3* follow the similar degrading tracks when manipulated features increase as displayed in Section 4.4.4: for *Defense1*, the weight evenness yields the classifier with more evenly-distributed feature weights, which in turn tends to cause the features to lose the significant information for classification; for *Defense2*, when more feature being manipulated in the malicious apps, the trained model more likely produces a distribution that is very close to that of the benign apps, which may not be able to differentiate benign and malicious apps accurately; for *Defense3*, the model is built on a carefully selected feature subset, whose robustness could be compromised when attackers manipulate a certain number of these features. *Defense4* ($T = 1$) utilizes the same gradient masking idea over label space to improve the robustness of the learning model, but soft labels in training have limited impact on the linear learning classifier with only two outputs. For *DroidEye* itself, small feature manipulations may not induce high output variation for the learning model, but after modifying a large number of features, the adversarial gradients may be significantly changed even in the continuous feature space, and thus its performance suffers from some drop-off, which performs slightly worse than *SecureDroid*.

4.6 Summary

In this section, we explore the adversarial attacks corresponding to the different scenarios, and define a general attack strategy to thoroughly assess the adversary behaviors. Resting on the learning-based classifier which is degraded by the adversarial

malware attacks, we propose three secure-learning paradigms *SecDefender*, *SecureDroid*, and *DroidEye* to counter these adversarial attacks. In our proposed methods, *SecDefender* is formulated against well-crafted attack *AdvAttack* through investigating the property of the feature set observed from the real sample collection, adopts classifier retraining technique, and enhances the robustness of the classifier using security regularization. *SecureDroid* is independent from the skills and capabilities of the attackers, and considers different importances of the features associated with their contributions to the classification problem and manipulation costs to the adversarial attacks; more specifically, in our developed system *SecureDroid*, a novel feature selection method *SecCLS* is proposed to reduce the possibility to select those features attackers tend to manipulate and thus helps to construct more secure classifier, and an ensemble learning approach *SecENS* is further proposed to aggregate the individual classifiers that are constructed using the proposed *SecCLS*. *DroidEye* thoroughly gets rid of empirical assumption for the adjustable parameters of the learning model, and improves the system security through feature space transformation, leaving model and training data unchanged.

Comprehensive experiments on the real sample collections from Comodo Cloud Security Center are conducted to validate the effectiveness of *SecDefender*, *SecureDroid*, and *DroidEye*. The results demonstrate that *SecDefender* can be resilient against attacks, but the limitation is relying on the skills and capacities of the attackers. For *SecureDroid*, the results demonstrate that our feature selection method *SecCLS* is more resilient to disrupt the feature manipulations, and *SecureDroid* can improve the security against various kinds of adversarial attacks even that attackers are with different skills and capabilities or have different knowledge about the targeted learning system, but the cons of *SecureDroid* lie in empirical assumption of feature manipulations and adjustable parameters. For *DroidEye*, the learning model can reduce the sensitivity to small feature manipulations, and preserve the reasonable generalization ability against the adversarial attacks. According to different application scenarios, these secure-learning models can be feasible in practical use for different malware detection tasks. The research work conducted in this chapter have been also published in the following papers:

- Lingwei Chen, Shifu Hou, Yanfang Ye ⊠. "SecureDroid: Enhancing Security of Machine Learning-based Detection against Adversarial Android Malware Attacks", ACSAC '17 Proceedings of the 33rd Annual Computer Security Applications Conference, 362–372, 2017.

- Lingwei Chen, Yanfang Ye ⊠, Thirimachos Bourlai. "Adversarial Machine Learn-

ing in Malware Detection: Arms Race between Evasion Attack and Defense“, EISIC ’17 European Intelligence and Security Informatics Conference, 99–106, 2017.

- Lingwei Chen, Yanfang Ye ✉. “SecMD: Make Machine Learning More Secure Against Adversarial Malware Attacks”, AI ’17 Australasian Joint Conference on Artificial Intelligence, 76–89, 2017.
- Lingwei Chen, Shifu Hou, Yanfang Ye ✉, Lifei Chen. “An Adversarial Machine Learning Model Against Android Malware Evasion Attacks”, Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data, 43–55, 2017.
- Lingwei Chen, Shifu Hou, Yanfang Ye ✉, Shouhuai Xu. “DroidEye: Fortifying Security of Learning-based Classifier against Adversarial Android Malware Attacks”, IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 782-789, 2018.

Chapter 5

Conclusion and Future Work

In this chapter, the summary of the contributions of this dissertation are presented. Then some future extensions of the current work are described.

5.1 Conclusion

Intelligent Malware Detection Utilizing file-to-file relations In intelligent malware detection, machine learning techniques exploit various classification approaches based on different feature representations to detect malicious files, which have set some successful examples for malware detection. However, such techniques mostly utilize local features either statically or dynamically extracted from file samples, while rarely investigating relations among file samples for malware detection. Recently, features beyond file content are starting to be leveraged for malware detection [149, 25, 121, 72], such as machine-to-file relations [25] and file-to-file relations (e.g., file co-existence) [149, 121], which provide invaluable insight about the properties of file samples [149]. In this dissertation, we take a further step to delve deeper into the relationship characteristics of malware and benign files, and investigate how we can construct the file-to-file relation graph between malware and benign file, what graph-based features, relationship characteristics, and representations can be employed for malware detection, and how we can build effective learning frameworks over graph for malware detection. The conclusion for intelligent malware detection using file-to-file relations can be summarized as follow:

- We provide deep analysis of file-to-file relations between malware and benign files and study how the file co-existence relation graphs can be constructed.
- Resting on the constructed file-to-file relation graphs, we design an enhanced Belief

Propagation algorithm for unknown file labeling that fine tunes various components used in the algorithm and formulates the new message and belief read-out functions.

- We investigate several new and robust graph-based features for malware detection and reveal the characteristics of file relations, based on which we propose an effective active learning framework (MSIA+EBP) for malware detection.
- We leverage a sequence modeling method Long Short-term Memory to learn the representations of files in our constructed graph which captures the long-range structural information.

To the best of our knowledge, this is the first investigation of the relationship characteristics for the file-to-file relations in malware detection using social network analysis, which yields great value and unveils a new avenue for better understanding malware's file relation ecosystem.

Enhancing Security of Learning-based Systems in Malware Detection As machine learning based detections become more widely deployed, the adversary incentives for defeating them increases. That is, machine learning itself may open the possibility for an adversary who actively manipulate the data to make the classifier produce errors. In this dissertation, we also present and study several topics to understand how we can define adversarial malware attacks, and how the security of a machine learning-based malware detection system can be enhanced in different scenarios. The conclusion for enhancing security of learning-based systems in malware detection can be summarized as follow:

- We explore the adversarial attacks corresponding to the different scenarios, thoroughly assess the adversary behaviors through feature manipulations, adversarial cost, and attack goals, and accordingly present a general attack strategy for further investigations. Resting on the learning-based classifier which is degraded by the adversarial malware attacks, we propose three secure-learning models *SecDefender*, *SecureDroid*, and *DroidEye* to counter these attacks.
- *SecDefender* adopts classifier retraining technique on basis of an attack model *AdvAttack* through investigating the property of the feature set observed from the real sample collection and their different contributions, and enhances the robustness of the classifier using the security regularization term.

- *SecureDroid* is independent from the skills and capabilities of the attackers, and considers different importances of the features associated with their contributions to the classification problem and manipulation costs to the adversarial attacks; more specifically, *SecCLS* is proposed to reduce the possibility to select those features attackers tend to manipulate and thus helps to construct more secure classifier, while *SecENS* is further proposed to aggregate the individual classifiers that are constructed using the proposed *SecCLS* to improve system security while not compromising detection accuracy.

- *DroidEye* thoroughly gets rid of empirical assumption for the adjustable parameters of the learning model, utilizes count featurization to transform the binary feature space into continuous probabilities encoding the distribution in each class to reduce the adversarial gradient of the learning model, and then introduces softmax with adversarial parameter to find the best trade-off between security and accuracy for the classifier.

Comprehensive experiments on the real sample collections from Comodo Cloud Security Center are conducted to validate the effectiveness of *SecDefender*, *SecureDroid*, and *DroidEye*. The results demonstrate that *SecDefender* can be resilient against attacks, but the limitation is relying on the skills and capacities of the attackers. For *SecureDroid*, the results demonstrate that our feature selection method *SecCLS* is more resilient to disrupt the feature manipulations, and *SecureDroid* can improve the security against the adversarial attacks even that attackers are with different skills and capabilities or have different knowledge about the targeted learning system. For *DroidEye*, the learning model can reduce the sensitivity to small feature manipulations, and preserve the reasonable generalization ability against the adversarial attacks. According to different application scenarios, these secure-learning models can be feasible in practical use for different malware detection tasks.

5.2 Future Work

In this section, we propose several future research topics based on our research goals. These future work are summarized as follows.

Intelligent malware detection using heterogeneous file contents and relations: Both PE files and Android apps can be characterized by a rich source of heterogeneous information, including their either static or dynamic content features (e.g., API calls, system

calls, dynamic behaviors, network traffic, etc.), and their relatedness over different types of relationships (e.g., co-author, co-affiliation, content sharing, temporal relations, etc.). We have presented effective graph inference, active learning and graph representation learning frameworks for malware detection based on the constructed file coexistence graphs, but barely considered different potential file relations and contents. To fully leverage such heterogeneous information, we would like to investigate how to use multi-view graphs, heterogeneous information network, or attributed network as an abstract representation to provide a natural way of expressing complex file relationships and semantics. Accordingly, we also want to elaborate innovative methods to learn the latent feature representations over these new graphs and networks to integrate both structural and semantic information for malware detection.

More defensive learning models in practical use for malware detection: According to different application scenarios, *SecDefender*, *SecureDroid*, and *DroidEye* all have provided significant solutions to enhance the security of machine learning-based classifier against adversarial attacks. But there are also some cons for these learning paradigms: *SecDefender* makes strong assumptions about the structure of the data and the attack model; *SecureDroid* empirically decides the adjustable parameters and manipulation costs, while feature manipulation methods (addition or elimination) are determined through the assumption that attackers conduct a well-crafted attack and are able to utilize information gain or max-relevance to calculate different contributions of the features for the classification of malicious and benign files respectively; *DroidEye* suffers from the performance drop-off when a large number of features being manipulated. Consequently, we want to weaken the assumption and construct more resilient solutions against the advanced attacks through analyzing learning model and feature space, and limiting the data exposure to adversarial attacks. In addition to evasion attacks, poisoning attacks and the corresponding secure solutions is also an important task for malware detection.

Gaining insight into the malware development and dissemination ecosystem: As malware has been progressively evolving into more complex threat, it's important to understand how the major players in the malware industry fit together and how these relationships affect the ways that malware is developed, distributed and ultimately used in attacks. The more we know about the activities of malware authors, malware distributors and malware affiliates at a large scale, the better we can prepare ourselves to defend against these attacks. We'd like to gain insight into the malware development and dissemination ecosystem to endeavor to gain a holistic and in-depth understanding about

the scope and magnitude malicious display, the features of their infrastructures, and the behaviors of malicious parties, and develop infrastructure-aware technologies to detect these malicious activities. To reveal these insights will help us decompose the relationships between cyber-criminals in malware industry, and accordingly facilitate securing the cyberspace.

List of Publications

1. Yanfang Ye ✉, Shifu Hou, Lingwei Chen, Xin Li, Liang Zhao, Shouhuai Xu, Jiabin Wang, Qi Xiong. "ICSD: An Automatic System for Insecure Code Snippet Detection in Stack Overflow over Heterogeneous Information Network", ACSAC '18 Proceedings of the 34rd Annual Computer Security Applications Conference, 542–552, 2018. **(20.1% acceptance rate)**

2. Yanfang Ye ✉, Shifu Hou, Lingwei Chen, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, Fudong Shao. "AiDroid: When Heterogeneous Information Network Marries Deep Neural Network for Real-time Android Malware Detection", arXiv preprint arXiv:1811.01027, 2018.

3. Lingwei Chen, Shifu Hou, Yanfang Ye ✉, Shouhuai Xu. "DroidEye: Fortifying Security of Learning-based Classifier against Adversarial Android Malware Attacks", IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), 782-789, 2018.

4. Yanfang Ye ✉, Lingwei Chen, Shifu Hou, William Hardy, Xin Li. "DeepAM: A Heterogeneous Deep Learning Framework for Intelligent Malware Detection", Knowledge and Information Systems (KAIS), Vol.54(2), 265–285, 2018. **(2019 Impact factor: 2.247)**

5. Lingwei Chen, Shifu Hou, Yanfang Ye ✉. "SecureDroid: Enhancing Security of Machine Learning-based Detection against Adversarial Android Malware Attacks", ACSAC '17 Proceedings of the 33rd Annual Computer Security Applications Conference, 362–372, 2017. **(19.7% acceptance rate)**

6. Lingwei Chen, Yanfang Ye ✉, Thirimachos Bourlai. "Adversarial Machine Learning in Malware Detection: Arms Race between Evasion Attack and Defense", EISIC '17 European Intelligence and Security Informatics Conference, 99–106, 2017. **(IEEE EISIC 2017 Best Paper Award)**

7. Lingwei Chen, Yanfang Ye ✉. "SecMD: Make Machine Learning More Secure

Against Adversarial Malware Attacks”, AI ’17 Australasian Joint Conference on Artificial Intelligence, 76–89, 2017.

8. Lingwei Chen, Shifu Hou, Yanfang Ye ✉, Lifei Chen. “An Adversarial Machine Learning Model Against Android Malware Evasion Attacks”, Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data, 43–55, 2017.

9. Shifu Hou, Lingwei Chen, Yanfang Ye ✉, Lifei Chen. “Deep Analysis and Utilization of Malware’s Social Relation Network for Its Detection”, Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data, 31–42, 2017.

10. Shifu Hou, Aaron Saas, Lingwei Chen, Yanfang Ye ✉, Thirimachos Bourlai. “Deep Neural Networks for Automatic Android Malware Detection”, ASONAM ’17 Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, 803–810, 2017.

11. William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye ✉, Xin Li. “DL4MD: A Deep Learning Framework for Intelligent Malware Detection”, DMIN ’16 International Conference on Data Mining, 61-67, 2016.

12. Abu HM Rubaiyat, Tanjin T Toma, Masoumeh Kalantari-Khandani, Syed A Rahman, Lingwei Chen, Yanfang Ye ✉, Christopher S Pan. “Automatic Detection of Helmet Uses for Construction Safety”, WIW ’16 IEEE/WIC/ACM International Conference on Web Intelligence Workshops, 135–142, 2016.

13. Lingwei Chen, William Hardy, Yanfang Ye ✉, Tao Li. “Analyzing File-to-File Relation Network in Malware Detection”, WISE ’15 International Conference on Web Information Systems Engineering, 415–430, 2015.

14. Lingwei Chen, Tao Li, Melih Abdulhayoglu, Yanfang Ye ✉. “Intelligent Malware Detection Based on File Relation Graphs”, ICSC ’15 IEEE International Conference on Sematic Computing, 85–92, 2015.

Bibliography

[1] M. S. Alam and S. T. Vuong, "Random forest classification for detecting android malware," in *GreenCom-iThings-CPSCom*, 2013, pp. 663–669.

[2] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in computer Virology*, vol. 7, no. 4, pp. 247–258, 2011.

[3] Android, "Application fundamentals," in *https://developer.android.com/guide/components/fundamentals.html*, 2017.

[4] AV-TEST, "Malware statistics," in *https://www.av-test.org/en/statistics/malware/*, 11 2018.

[5] D. Babic, D. J. Klein, I. Lukovits, S. Nikolic, and N. Trinajstic, "Resistance-distance matrix: A computational algorithm and its application," *International Journal of Quantum Chemistry*, vol. 90, no. 1, pp. 166–176, 2002.

[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[7] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *International Workshop on Recent Advances in Intrusion Detection*, 2007, pp. 178–197.

[8] ——, "Automated classification and analysis of internet malware," in *International Workshop on Recent Advances in Intrusion Detection.* Springer, 2007, pp. 178–197.

[9] U. Baldangombo, N. Jambaljav, and S.-J. Horng, "A static malware detection system using data mining methods," *CoRR Journal*, vol. 1308, no. 2831, 2013.

[10] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[11] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006, pp. 16–25.

[12] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology*, vol. 2, no. 1, pp. 67–77, 2006.

[13] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, “Evasion attacks against machine learning at test time,” in *Joint European conference on machine learning and knowledge discovery in databases (ECML PKDD)*, 2013, pp. 387–402.

[14] B. Biggio, G. Fumera, and F. Roli, “Security evaluation of pattern classifiers under attack,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 984–996, 2014.

[15] B. Biggio, F. Roli, and G. Fumera, “Design of robust classifiers for adversarial environments,” in *Proceedings of IEEE International Conference on SMC*, 2011, pp. 977–982.

[16] B. Biggio, G. Fumera, and F. Roli, “Multiple classifier systems for robust classifier design in adversarial environments,” *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1, pp. 27–41, 2010.

[17] S. M. Bridges, R. B. Vaughn *et al.*, “Fuzzy data mining and genetic algorithms applied to intrusion detection,” in *Proceedings of 12th Annual Canadian Information Technology Security Symposium*, 2000, pp. 109–122.

[18] M. Bruckner, C. Kanzow, and T. Scheffer, “Static prediction games for adversarial learning problems,” *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 2617–2654, 2012.

[19] M. Bruckner and T. Scheffer, “Stackelberg games for adversarial prediction problems,” in *KDD ’11 Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 547–555.

[20] C. Cade, “Understanding heuristic-based scanning vs. sandboxing,” in *https://www.opswat.com/blog/understanding-heuristic-based-scanning-vs-sandboxing*, 2015.

[21] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[22] N. Cesa-Bianchi, C. Gentile, F. Vitale, and G. Zappella, “Active learning on trees and graphs,” in *COLT*, 2013, pp. 320—-332.

[23] S. Cesare, Y. Xiang, and W. Zhou, “Control flow-based malware variant detection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 4, pp. 307–317, 2014.

[24] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[25] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, “Polonium: Tera-scale graph mining for malware detection,” in *Proceedings of the 2011 SIAM International Conference on Data Mining*, 2011, pp. 131–142.

[26] K. Chen, P. Zhu, and Y. Xiong, “Mining spam accounts with user influence,” in *International Conference on Information Science and Cloud Computing Companion (ISCC-C)*, 2013, pp. 167–173.

[27] L. Chen, W. Hardy, Y. Ye, and T. Li, "Analyzing file-to-file relation network in malware detection," in *WISE '15 International Conference on Web Information Systems Engineering*, 2015, pp. 415–430.

[28] L. Chen, S. Hou, and Y. Ye, "Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *ACSAC '17 Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 362–372.

[29] L. Chen, S. Hou, Y. Ye, and L. Chen, "An adversarial machine learning model against android malware evasion attacks," in *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, 2017, pp. 43–55.

[30] L. Chen, S. Hou, Y. Ye, and S. Xu, "Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks," in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM).* IEEE, 2018, pp. 782–789.

[31] L. Chen, T. Li, M. Abdulhayoglu, and Y. Ye, "Intelligent malware detection based on file relation graphs," in *ICSC '15 IEEE International Conference on Sematic Computing*, 2015, pp. 85–92.

[32] L. Chen and Y. Ye, "Secmd: Make machine learning more secure against adversarial malware attacks," in *AI '17 Australasian Joint Conference on Artificial Intelligence*, 2017, pp. 76–89.

[33] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *EISIC '17 European Intelligence and Security Informatics Conference*, 2017, pp. 99–106.

[34] T. Chen and J.-M. Robert, *Statistical Methods in Computer Security.* CRC Press, 2004.

[35] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[36] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Security and Privacy, 2005 IEEE Symposium on.* IEEE, 2005, pp. 32–46.

[37] Cybersecurity-Ventures, "Cybercrime damages are predicted to cost the world $6 trillion annually by 2021," in *https://www.prnewswire.com/news-releases/cybercrime-damages-are-predicted-to-cost-the-world-6-trillion-annually-by-2021-300540158.html*, 10 2017.

[38] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *KDD '04 Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 99–108.

[39] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 2, pp. 289–302, 2016.

[40] D. Debarr, H. Sun, and H. Wechsler, "Adversarial spam detection using the randomized hough transform-support vector machine," in *ICMLA '13 12th international conference on Machine Learning and Applications (ICMLA)*, 2013, pp. 299–304.

[41] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[42] R. Diestel, *Graph Theory, Vol. 173, 4th Edition.* Heidelberg: Springer, 2010.

[43] T. G. Dietterich, "Ensemble methods in machine learning," *Multiple Classifier Systems*, vol. 1, pp. 1–15, 2000.

[44] R. A. Dunne, *A Statistical Approach to Neural Networks for Pattern Recognition.* Wiley-Interscience, 1st edition, 2007.

[45] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware analysis techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, pp. 6:1–6:42, 2008.

[46] ——, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, p. 6, 2012.

[47] Y. Fan, S. Hou, Y. Zhang, Y. Ye, and M. Abdulhayoglu, "Gotcha-sly malware!: Scorpion a metagraph2vec based malware detection system," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* ACM, 2018, pp. 253–262.

[48] E. Filiol, "Malware pattern scanning schemes secure against blackbox analysis," *Journal in Computer Virology*, vol. 2, no. 1, pp. 35–50, 2006.

[49] E. Filiol, G. Jacob, and M. Liard, "Evaluation methodology and theoretical model for antiviral behavioural detection strategies," *Journal in Computer Virology*, vol. 3, no. 1, pp. 23–37, 2007.

[50] I. Firdausi, A. Erwin, A. S. Nugroho *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on.* IEEE, 2010, pp. 201–203.

[51] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *Journal of Information Security*, vol. 5, no. 02, p. 56, 2014.

[52] GDATA, "History of malware," in *https://www.gdata-software.com/seccurity-labs/information/history-of-malware*, 2005.

[53] A. Globerson and S. Roweis, "Nightmare at test time: Robust learning by feature deletion," in *ICML '06 Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 353–360.

[54] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR '15*, 2015.

[55] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[56] R. A. Grimes, *Malicious mobile code: Virus protection for Windows.* " O'Reilly Media, Inc.", 2001.

[57] H. Guo, Y. Li, Y. Li, X. Liu, and J. Li, "Bpso-adaboost-knn ensemble learning algorithm for multi-class imbalanced data classification," *Engineering Applications of Artificial Intelligence*, vol. 49, pp. 176–193, 2016.

[58] N. Haghtalab, F. Fang, T. H. Nguyen, A. Sinha, A. D. Procaccia, and M. Tambe, "Three strategies to success: Learning adversary models in security games," in *IJCAI'16*, 2016, pp. 308–314.

[59] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques 3rd Edition.* Waltham, MA, USA: Morgan Kaufmann, 2011.

[60] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "Dl4md: A deep learning framework for intelligent malware detection," in *DMIN '16 International Conference on Data Mining*, 2016, pp. 61–67.

[61] V. Harrison and J. Pagliery, "Nearly 1 million new malware threats released every day," in *http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security*, 2015.

[62] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[63] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[64] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[65] S. Hou, L. Chen, E. Tas, I. Demihovskiy, and Y. Ye, "Cluster-oriented ensemble classifiers for malware detection," in *IEEE International Conference on Sematic Computing (IEEE ICSC)*, 2015, pp. 189–196.

[66] S. Hou, L. Chen, Y. Ye, and L. Chen, "Deep analysis and utilization of malware's social relation network for its detection," in *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, 2017, pp. 31–42.

[67] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs," in *WIW '16 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*, 2016.

[68] S. Hou, A. Saas, Y. Ye, and L. Chen, "Droiddelver: An android malware detection system using deep belief network based on api call blocks," in *WAIM '16 International Conference on Web-Age Information Management*, 2016, pp. 54–66.

[69] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *KDD '17 Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1507–1515.

[70] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue University*, vol. 48, 2007.

[71] W. Jung, S. Kim, and S. Choi, "Deep learning for zero-day flash malware detection," in *36th IEEE Symposium on Security and Privacy*, 2015.

[72] N. Karampatziakis, J. W. Stokes, A. Thomas, and M. Marinescu, "Using file relationships in malware classification," in *DIMVA 2012: Detection of Intrusions and Malware, and Vulnerability Assessment*, 2012, pp. 1–20.

[73] Kaspersky, "The great bank robbery," in *http://www.kaspersky.com/about/news/virus/2015/Carbanak-cybergang-steals-1-bn-USD-from-100-financial-institutions-worldwide*, 2015.

[74] KasperskyLab, "4 in 5 malware attacks cause problems for users and 1 in 3 result in money loss," in *http://www.kaspersky.com/about/news/virus/2015/4-in-5-Malware-Attacks-Cause-Problems-for-Users-and-1-in-3-Result-in-Money-Loss*, 2015.

[75] J. Kephart and W. Arnold, "Automatic extraction of computer virus signatures," in *Proceedings of the 4th Virus Bulletin International Conference*, 1994, pp. 178–184.

[76] Kingsoft, "2015-2016 internet security research report in china," in *http://cn.cmcm.com/news/media/2016-01-14/60.html*, 2016.

[77] C. Kolbitsch, E. Kirda, and C. Kruegel, "The power of procrastination: detection and mitigation of execution-stalling malicious code," in *CCS '11*, 2011, pp. 285–296.

[78] A. Kolcz and C. H. Teo, "Feature weighting for improved classifier robustness," in *CEAS '09 Sixth conference on email and anti-spam*, 2009.

[79] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *KDD '04*, 2004, pp. 470–478.

[80] D. Kong and G. Yan, "Discriminant malware distance learning on structural information for automated malware classification," in *KDD '13 Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1357–1365.

[81] M. Lecuyer, R. Spahn, R. Geambasu, T.-K. Huang, and S. Sen, "Pyramid: Enhancing selectivity in big data protection with count featurization," in *Security and Privacy (SP), 2017 IEEE Symposium on.* IEEE, 2017, pp. 78–95.

[82] B. Li and Y. Vorobeychik, "Feature cross-substitution in adversarial classification," in *NIPS'14*, 2014, pp. 2087–2095.

[83] B. Li, Y. Vorobeychik, and X. Chen, "A general retraining framework for scalable adversarial classification," in *NIPS 2016 Workshop on Adversarial Training*, 2016.

[84] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," *International Journal of Security and Its Applications*, vol. 9, no. 5, pp. 205–216, 2015.

[85] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[86] J. Liu, Z. He, L. Wei, and Y. Huang, "Content to node: Self-translation network embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* ACM, 2018, pp. 1794–1802.

[87] D. Lowd and C. Meek, "Adversarial learning," in *KDD '05*, 2005, pp. 641–647.

[88] M. Ludwig and D. Noah, *The giant black book of computer viruses.* American Eagle Books, 2017.

[89] J. Mar, I.-F. Hsiao, Y. C. Yeh, C.-C. Kuo, and S.-R. Wu, "Intelligent intrusion detection and robust null defense for wireless networks," *International Journal of Innovative Computing Information and Control*, vol. 8, no. 5, pp. 3341–59, 2012.

[90] M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "Cloud-based malware detection for evolving data streams," *ACM TMIS*, vol. 2, no. 3, pp. 16:1–16:27, 2011.

[91] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of network and computer applications*, vol. 36, no. 1, pp. 42–57, 2013.

[92] R. Moskovitch, C. Feher, and Y. Elovici, "A chronological evaluation of unknown malcode detection," *LNCS: Intelligence and Security Informatics*, vol. 5477, no. 2009, pp. 112–117, 2009.

[93] I. Muslea, S. Minton, and C. A. Knoblock, "Active learning with multiple views," *Journal of Artificial Intelligence Research*, vol. 27, pp. 203–233, 2006.

[94] H. T. Nguyen and A. Smeulders, "Active learning using pre-clustering," in *Proceedings of the twenty-first international conference on Machine learning.* ACM, 2004, p. 79.

[95] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, "Novel active learning methods for enhanced pc malware detection in windows os," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5843–5857, 2014.

[96] A. Nøkland, "Improving back-propagation by adding an adversarial gradient," *arXiv preprint arXiv:1510.04189*, 2015.

[97] N. Noorshams and M. J. Wainwright, "Belief propagation for continuous state spaces: Stochastic message-passing with quantitative guarantees," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 2799–2835, 2013.

[98] J. Ouellette, A. Pfeffer, and A. Lakhotia, "Countering malware evolution using cloud-based learning," in *8th International Conference on Malicious and Unwanted Software (MALWARE)*, 2013, pp. 85–94.

[99] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

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

[101] Y. Park, Q. Zhang, D. Reeves, and V. Mulukutla, "Antibot: Clustering common semantic patterns for bot detection," in *COMPSAC '10*, 2010, pp. 262–272.

[102] J. Pearl, *Reverend Bayes on inference engines: A distributed hierarchical approach.* Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

[103] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy," *IEEE TPAMI*, vol. 27, no. 8, pp. 1226–1238, 2005.

[104] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2014, pp. 701–710.

[105] B. Rankin, "A brief history of malware — its evolution and impact," in *https://www.lastline.com/blog/history-of-malware-its-evolution-and-impact/*, 4 2018.

[106] J. Raymond, "Malware definition and their removal methods," in *https://antivirus.comodo.com/blog/comodo-news/malware-definition-and-their-removal/*, 7 2018.

[107] R. Rehmani, G. C. Hazarika, and G. Chetia, "Malware threats and mitigation strategies: A survey," *Journal of Theoretical and Applied Information Technology*, vol. 29, no. 2, pp. 69–73, 2011.

[108] F. Roli, B. Biggio, and G. Fumera, "Pattern recognition systems under attack," in *CIARP '13*, 2013, pp. 1–8.

[109] I. A. Saeed, A. Selamat, and A. M. Abuagoub, "A survey on malware and malware detection systems," *International Journal of Computer Applications*, vol. 67, no. 16, 2013.

[110] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in *International Symposium on Distributed Computing and Artificial Intelligence*, 2011, pp. 415–422.

[111] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions.* Springer, 2013, pp. 271–280.

[112] I. Santos, C. Laorden, and P. G. Bringas, "Collective classification for unknown malware detection," in *Security and Cryptography (SECRYPT), 2011 Proceedings of the International Conference on.* IEEE, 2011, pp. 251–256.

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

[114] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *SP '01*, 2001, p. 38.

[115] J. Scott, *Social Networks Analysis: A Hand Book, 2nd Edition.* SAGE Publications Ltd, 2000.

[116] S. Shah, H. Jani, S. Shetty, and K. Bhowmick, "Virus detection using artificial neural networks," *International Journal of Computer Applications*, vol. 84, no. 5, pp. 17–23, 2013.

[117] A. Shrivastava, A. C. Konig, and M. Bilenko, "Time adaptive sketches (ada-sketches) for summarizing data streams," in *Proceedings of the 2016 International Conference on Management of Data.* ACM, 2016, pp. 1417–1432.

[118] F. Stroud, "Cryptomining malware," in *https://www.webopedia.com/TERM/C/cryptomining-malware.html*, 2018.

[119] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static analyzer of vicious executables (save)," in *ACSAC '04*, 2004, pp. 326–334.

[120] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[121] A. Tamersoy, K. Roundy, and D. H. Chau, "Guilt by association: large scale malware detection by mining file-relation graphs," in *KDD '14 Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1524–1533.

[122] R. Thomas and M. Ligh, "Method and system for automatic detection and analysis of malware," Jan. 26 2016, uS Patent 9,245,114.

[123] A. Vasudevan and R. Yerraballi, "Spike: engineering malware analysis tools using unobtrusive binary-instrumentation," in *Proceedings of the 29th Australasian Computer Science Conference-Volume 48*. Australian Computer Society, Inc., 2006, pp. 311–320.

[124] D. Venugopal and G. Hu, "Efficient signature based malware detection on mobile devices," *Mobile Information Systems*, vol. 4, no. 1, pp. 33–49, 2008.

[125] J. Von Neumann, "Theory and organization of complicated automata," *Burks (1966)*, pp. 29–87, 1949.

[126] N. Šrndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *SP '14*, 2014, pp. 197–211.

[127] F. Wang, W. Liu, and S. Chawla, "On sparse feature attacks in adversarial learning," in *ICDM '14*, 2014, pp. 1013–1018.

[128] T.-Y. Wang, S.-J. Horng, M.-Y. Su, C.-H. Wu, P.-C. Wang, and W.-Z. Su, "A surveillance spyware detection system based on data mining methods," in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. IEEE, 2006, pp. 3236–3241.

[129] Wikipedia, "Timeline of computer viruses and worms," in *https://en.wikipedia.org/wiki/Timeline_of_computer_viruses_and_worms*, 11 2018.

[130] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[131] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *ASIAJCIS '12 Proceedings of the 2012 Seventh Asia Joint Conference on Information Security*, 2012.

[132] W.-C. Wu and S.-H. Hung, "Droiddolphin: a dynamic android malware detection framework using big data and machine learning," in *RACS '14 Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, 2014.

[133] Y. Wu, T. Ren, and L. Mu, "Importance reweighting using adversarial-collaborative training," in *NIPS 2016 Workshop*, 2016.

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

[135] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers a case study on pdf malware classifiers," in *NDSS '16*, 2016.

[136] C. Yang, R. Harkreader, J. Zhang, S. Shin, and G. Gu, "Analyzing spammer's social networks for fun and profit: A case study of cyber criminal ecosystem on twitter," in *Proceedings of the 21st international conference on World Wide Web (WWW '12)*, 2012, pp. 71–80.

[137] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications," *ESORICS European Symposium on Research in Computer Security*, vol. 8712, pp. 163–182, 2014.

[138] P. Yang and P. Zhao, "A min-max optimization framework for online graph classification," in *CIKM '15*, 2015, pp. 643–652.

[139] Y. Ye, D. Wang, T. Li, and D. Ye, "Imds: Intelligent malware detection system," in *KDD '07*, 2007, pp. 1043–1047.

[140] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent pe-malware detection system based on association mining," *Journal in Computer Virology*, vol. 4, no. 4, pp. 323–334, 2008.

[141] Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, and M. Zhao, "Sbmds: an interpretable string based malware detection system using svm ensemble with bagging," *Journal in Computer Virology*, vol. 5, pp. 283–293, 2009.

[142] Y. Ye, L. Chen, S. Hou, W. Hardy, and X. Li, "Deepam: a heterogeneous deep learning framework for intelligent malware detection," *Knowledge and Information Systems*, vol. 54, no. 2, pp. 265–285, 2018.

[143] Y. Ye, S. Hou, L. Chen, J. Lei, W. Wan, J. Wang, Q. Xiong, and F. Shao, "Aidroid: When heterogeneous information network marries deep neural network for real-time android malware detection," *arXiv preprint arXiv:1811.01027*, 2018.

[144] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, p. 41, 2017.

[145] Y. Ye, T. Li, Y. Chen, and Q. Jiang, "Automatic malware categorization using cluster ensemble," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2010, pp. 95–104.

[146] Y. Ye, T. Li, K. Huang, Q. Jiang, and Y. Chen, "Hierarchical associative classifier (hac) for malware detection from the large and imbalanced gray list," *Journal of Intelligent Information Systems*, vol. 35, no. 1, pp. 1–20, 2010.

[147] Y. Ye, T. Li, Q. Jiang, Z. Han, and L. Wan, "Intelligent file scoring system for malware detection from the gray list," in *KDD '09*, 2009, pp. 1385–1394.

[148] Y. Ye, T. Li, Q. Jiang, and Y. Wang, "Cimds: adapting postprocessing techniques of associative classification for malware detection," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 3, pp. 298–307, 2010.

[149] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu, "Combining file content and file relations for cloud based malware detection," in *KDD '11 Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 222–230.

[150] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," in *Exploring artificial intelligence in the new millennium*, 2003, pp. 239–269.

[151] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," in *SIGCOMM '14 Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 371–372.

[152] F. Zhang, P. P. K. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 766–777, 2015.

[153] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "Antimaldroid: An efficient svm-based malware detection framework for android," in *ICICA '11 International Conference on Information Computing and Applications*, 2011, pp. 158–166.