UNIVERSITY OF PIRAEUS

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES

DEPARTMENT OF DIGITAL SYSTEMS

Implementing AI-driven Methodologies for Cyberattack Detection

Doctoral Thesis

Panagiotis Bountakas

Piraeus, 2023

UNIVERSITY OF PIRAEUS

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGIES

DEPARTMENT OF DIGITAL SYSTEMS

Implementing AI-driven Methodologies for Cyberattack Detection

Supervisor: Prof. Christos Xenakis

Advisors: Prof. Costas Lambrinoudakis,
Prof. Dimosthenis Kyriazis

This thesis is submitted for the degree of

Doctor of Philosophy

Piraeus, 2023

Advisory Committee

Christos Xenakis	*Professor*, School of Information and Communication Technologies, Department of Digital Systems, University of Piraeus (*supervisor*)
Costas Lambrinoudakis	*Professor*, School of Information and Communication Technologies, Department of Digital Systems, University of Piraeus
Dimosthenis Kyriazis	*Professor*, School of Information and Communication Technologies, Department of Digital Systems, University of Piraeus

Examination Committee

Christos Xenakis	*Professor*, School of Information and Communication Technologies, Department of Digital Systems, University of Piraeus (*supervisor*)
Costas Lambrinoudakis	*Professor*, School of Information and Communication Technologies, Department of Digital Systems, University of Piraeus
Dimosthenis Kyriazis	*Professor*, School of Information and Communication Technologies, Department of Digital Systems, University of Piraeus
Sokratis Katsikas	*Professor*, Department of Information Security and Communication Technology, Norwegian University of Science and Technology
Stefanos Gritzalis	*Professor*, School of Information and Communication Technologies, Department of Digital Systems, University of Piraeus
Christos Kalloniatis	*Professor*, Department of Cultural Technology and Communication, University of the Aegean
Dionysis Xenakis	*Assistant Professor,* Department of Digital Industry Technologies, National and Kapodistrian University of Athens

To Hrys, my parents, brother and sister
for their invaluable support.

Acknowledgements

The journey of my PhD studies finished and along the way there are many people that assisted me accomplishing my goals. First, I would like to express my gratitude to my supervisor Professor Christos Xenakis for his invaluable mentoring, support, and guidance during my PhD studies. I would also like to thank the advisory committee, Professor Costas Lambrinoudakis and Professor Dimosthenis Kyriazis for their valuable guidance.

Moreover, I would like to express my appreciation to Assistant Professor Christoforos Dadoyan for his assistance in the early stages of my PhD studies. I would also like to extend my gratitude to Mr. Aristeidi Farao, Mr. Vaio Bolgoura, Mrs. Eleni Veroni, Mrs. Anna Angelogianni, and all the members of the Secure Systems Laboratory for valuable support and constructive discussions the past years.

Last but not least, I would like to thank my fiancé Hrys and my family for always believing in me and supporting my decisions. This dissertation could not be possible without you!

Abstract

The wide expansion of digital technologies and web applications has made daily activities easier and more amusing; however, it has led to the creation of new cybercriminal actions as well as the development of sophisticated cyberattacks. This new paradigm has made traditional defense solutions incapable of tackling the growth of cyberattacks and thus new defenses has appeared that exploit the efficacy of Artificial Intelligence systems.

Even though Machine Learning has often been used to detect and mitigate cyberattacks, the literature has several gaps and limitations that either render the current solutions unreliable or difficult to be deployed in real-life. Towards this direction and in order to address the limitations of existing works, this thesis has studied the security of Artificial Intelligence systems, surveying the current defense solutions against Adversarial Machine Learning attacks and has created a taxonomy of the identified defense solutions to facilitate researchers propose new robust defenses in the future. After identifying that Artificial Intelligence systems can be protected, this thesis focused on designing and developing Machine Learning defense methodologies against two well-known cyberattacks, known as phishing and exploit kits. The experimental results showed that the proposed methodologies, overall, improved the detection performance of phishing and exploit kit attacks, concluding that Machine Learning can be a useful weapon to stop the on-going threat of cyberattacks.

Furthermore, this thesis also studies a newly introduced code injection attack, named Server-side JavaScript Injection and proposed a methodology that has been created also as a software tool that automatically detects and exploits Server-side JavaScript Injection vulnerabilities.

Finally, this thesis has identified several directions for future research that hopefully will facilitate researchers in the future to create robust defense solutions and effectively tackle sophisticated cyberattacks.

Περίληψη

Η ευρεία επέκταση των ψηφιακών τεχνολογιών και των διαδικτυακών εφαρμογών έχει κάνει τις καθημερινές δραστηριότητες ευκολότερες και πιο διασκεδαστικές. Ωστόσο, οδήγησε στη δημιουργία νέων κυβερνοεγκληματικών ενεργειών καθώς και στην ανάπτυξη εξελιγμένων κυβερνοεπιθέσεων. Αυτό το νέο παράδειγμα έχει κάνει τις παραδοσιακές αμυντικές λύσεις ανίκανες να αντιμετωπίσουν την ανάπτυξη των κυβερνοεπιθέσεων και έτσι έχουν εμφανιστεί νέες άμυνες που εκμεταλλεύονται την αποτελεσματικότητα των συστημάτων Τεχνητής Νοημοσύνης.

Παρόλο που η Μηχανική Μάθηση έχει χρησιμοποιηθεί συχνά για τον εντοπισμό και τον μετριασμό των επιθέσεων στον κυβερνοχώρο, η βιβλιογραφία έχει αρκετά κενά και περιορισμούς που είτε καθιστούν τις τρέχουσες λύσεις αναξιόπιστες είτε δύσκολες στην εφαρμογή τους στην πραγματική ζωή. Προς αυτή την κατεύθυνση και προκειμένου να αντιμετωπιστούν οι περιορισμοί των υφιστάμενων ερευνών, η παρούσα διατριβή έχει μελετήσει την ασφάλεια των συστημάτων Τεχνητής Νοημοσύνης, ερευνώντας τις τρέχουσες αμυντικές λύσεις έναντι των επιθέσεων Adversarial Machine Learning και έχει δημιουργήσει μια ταξινόμηση των προσδιορισμένων αμυντικών λύσεων για να διευκολύνει τους ερευνητές να προτείνουν στο μέλλον νέες ισχυρές άμυνες. Αφού αναγνωρίσαμε ότι τα συστήματα Τεχνητής Νοημοσύνης μπορούν να προστατευτούν, αυτή η διατριβή επικεντρώθηκε στο σχεδιασμό και την ανάπτυξη αμυντικών μεθοδολογιών Μηχανικής Μάθησης έναντι δύο γνωστών κυβερνοεπιθέσεων, γνωστών ως phishing και exploit kit. Τα πειραματικά αποτελέσματα έδειξαν ότι οι προτεινόμενες μεθοδολογίες, συνολικά, βελτίωσαν την απόδοση ανίχνευσης επιθέσεων phishing και exploit kit, καταλήγοντας στο συμπέρασμα ότι η Μηχανική Μάθηση μπορεί να χρησιμοποιηθεί για να σταματήσει τη συνεχιζόμενη απειλή των κυβερνοεπιθέσεων.

Επιπλέον, η παρούσα διατριβή μελετά μια επίθεση code injection που εισήχθη πρόσφατα, η οποία ονομάζεται Server-side JavaScript Injection και προτείνει μια μεθοδολογία που έχει δημιουργηθεί επίσης ως εργαλείο λογισμικού που εντοπίζει και εκμεταλλεύεται αυτόματα ευπάθειες που σχετίζονται με τις επιθέσεις Server-side JavaScript Injection.

Τέλος, αυτή η διατριβή έχει εντοπίσει διάφορες κατευθύνσεις για μελλοντική έρευνα που ελπίζουμε ότι θα διευκολύνουν τους ερευνητές στο μέλλον να δημιουργήσουν ισχυρές αμυντικές λύσεις και να αντιμετωπίσουν αποτελεσματικά εξελιγμένες κυβερνοεπιθέσεις.

Table of Contents

List of Figures

List of Tables

1. Introduction

The wide expansion of digital technologies has resulted in an outburst of web applications and processes aiming to facilitate people in their in day-to-day activities. Varying from online shopping to remote monitoring of implantable medical devices, web applications are heavily used worldwide [1]. This evolution has introduced new ways for easier and cost-effective methods to benefit developers and IT professionals by providing them all the necessary tools to develop innovative applications. Furthermore, nowadays more and more organizations adopt the paradigm of remote working allowing their employees to work from anywhere (i.e., outside the protected organization's network) using their unsafe personal computers to access the organization's sensitive data. This shift contributed to a wider attack surface for cybercriminals that constantly search new innovative ways to compromise individuals' or organizations' security and privacy. In addition, the proliferation of Artificial Intelligence (AI) not only has led to the adoption of Machine Learning (ML) and Deep Learning (DL) methods in every technological domain, but also when it comes to cybersecurity, they have been proved a valuable weapon to security professionals' arsenal. Currently, ML and DL methods have replaced traditional security practices that were deployed for decades to mitigate cyberattacks, such as signature-based intrusion detection and regular expression rules and have been applied almost to every cybersecurity domain (e.g., biometric authentication, intrusion and anomaly behavior detection, etc.) advancing the results of traditional non-AI solutions. Moreover, the evolution of AI methods has introduced a new threat landscape, where adversaries aiming to trick ML and DL models by providing deceptive inputs to affect their performance (e.g., altering their output) acknowledged as Adversarial Machine Learning (AML).

The expansion of digital technologies apart from being beneficial for individuals and organizations, it has been also facilitated cybercriminals to come up with new innovative strategies or enhance existing ones to evade security solutions and perform their malicious actions undetected. Thus, this thesis focuses on the exploitation of AI methods to analyze and detect prominent cyberattacks as well as on enhancing the robustness of AI systems. More specifically, a survey on AML attacks has been conducted focusing mostly on existing defense methods that are designed either to enhance the robustness of ML/DL models against malicious actions or detect AML attacks. The survey resulted in a taxonomy that includes all the different defense methods that exist in the literature. In addition, the gaps of the literature are identified, and a discussion is performed providing future research directions to address this ongoing threat (see Chapter 2 for more information). Later on, three different types of cyberattacks have been studied and three different defenses have been developed as a mitigation to tackle these cyberattacks. In the beginning, even though phishing attacks are a common cybercriminal strategy for almost two decades, the last couple of years they have been highly evolved and as a result existing AI and non-AI solutions have failed to address them. Based on this observation, this thesis investigates the most known type of phishing attacks, namely phishing emails (see Chapter 3 for more information) and proposes an innovative Ensemble Learning methodology that can effectively detect phishing emails. Next, the second cyberattack that has been examined in this thesis is the relatively new threat of Exploit Kits (EKs); a sophisticated cybercriminal strategy that introduced the concept of Exploit-as-a-Service, where users with basic computer knowledge can rent access to EK servers and via their user friendly environment contaminate targeted computers using a plethora of malwares, including but not limited to ransomwares, trojans, and coin miners (see Chapter 4 for more information). As a defense solution against EKs, this thesis presented EKnad, which is a novel ML-based methodology for EKs detection via the network traces they leave behind during their infection procedure. Finally, a code injection attack was studied, named Server-side JavaScript Injection attack (SSJI). SSJI is a newly created cyberattack that affects applications (e.g., Node.js) that allow developers to execute code in the server-side (see Chapter 5 for more information). As a defense solution this thesis introduces a methodology, developed also as a software tool that can automatically detect and exploit SSJI vulnerabilities.

1.1 Motivation

The motivation of this thesis stems from various factors in the cybersecurity domain. These factors can be mainly classified into four categories: a) Limited research on the defense methods to tackle AML attacks; b) Evolution of phishing email attacks; c) Severity of EK attacks; d) Lack of detection methods for SSJI attacks.

First, even though AI is heavily applied to critical tasks, the security of AI systems takes a back seat when developing and evaluating ML/DL models. These models are usually developed based on the assumption that both the trained datasets are trustworthy, and the evaluating environment is secure. Hence, most AI systems are vulnerable to various attacks that aim to impact the algorithms' decisions by modifying the training data (known as poisoning attacks) or altering the output of the model (known as evasion attacks). Current research delves into attacking ML/DL models rather than defending them. More specifically, existing works mainly focus on the computer vision domain by introducing various methods for the generation of adversarial samples, which, when injected in the testing phase, can reduce the algorithms' performance significantly [24]. Nevertheless, novel AML attacks have been recently expanded also in other domains, including health, audio, Natural Language Processing (NLP), and cybersecurity, as well as in various algorithms (e.g., Convolutional Neural Networks, Random Forest, and Support Vector Machines), indicating the severity of AML [25], [26]. Furthermore, most of the current surveys in the field of AML focus only on a specific domain (e.g., computer vision or cybersecurity) and elaborate on the attacks against AI systems without delving into the possible defenses. Moreover, several articles in the literature highlight that the research on the defense solutions against AML attacks is still in its infancy and further research should be conducted to tackle the growth of AML attacks [27], [28], [29]. Adding to this, none of the existing survey articles solely elaborate on the existing detection and mitigation methods, leaving a knowledge gap for researchers wishing to develop new defense methods to confront AML. This leads to the first research question:

RQ1: ***Is it possible to create a taxonomy of existing defenses against AML attacks?***

Second, despite the longevity of phishing email attacks, traditional email filtering techniques have been inefficient in effectively confronting phishing email attacks. This can be validated by the latest phishing reports which highlight that in 2020 75% of organizations faced a phishing attack, as well as that 96% of phishing attacks were performed via emails [7], while the email threats rose by 64% compared to 2019 [8]. Moreover, phishing attacks account for 90% of data breaches provoking high financial losses of the order of $3.86 million [9]. In 2020, the COVID-19 pandemic cultivated a breeding ground for email attacks since between February 2020 and March 2020 phishing emails were raised by 667% [10]. More specifically, cyber-criminals exploit people's desperation (regarding the pandemic, vaccination, etc.) to deceive them into clicking on malicious links or attachments [11]. Also, in the last few years, the involvement of Machine Learning (ML) in the generation process of phishing attacks has led to more sophisticated and identical to benign phishing emails that can evade current detection solutions [12]. Moreover, on the one hand, numerous works exist in the literature that aiming to tackle phishing emails using innovative methods such as ML, DL, and Natural Language Processing (NLP). Yet, other novel methods, like Ensemble Learning (e.g., the combination of two or more ML algorithms) has progressed the results in several fields of research, such as phishing web page detection [13] and fraud detection [14]; however, it has not been comprehensively examined in the field of phishing email detection. On the other hand, several recent studies (e.g., [15], [16], [17]) have identified various gaps and limitations in the literature; to name a few: i) a narrow set of classifiers is usually considered, ii) balanced datasets are often employed, even though phishing detection is an imbalanced classification problem, iii) inadequate evaluation metrics, iv) dataset issues (i.e., limited available data sources for researchers), and v) obsolete phishing emails. This leads to the second research question:

RQ2: ***Can the combination of Ensemble Learning with hybrid features, overall, enhance the phishing email detection performance?***

Third, recent, large-scale study of EKs functionalities presented in [18] concluded that a common trait of EKs is their ability to quickly adapt to new malware trends. Indeed, nowadays EKs have evolved and shifted their modus operandi to more lucrative methods by delivering ransomware. For instance, the Maze ransomware[1] was initially delivered through spam email campaigns, but cybercriminals changed their tactics by taking advantage of EKs to infect computers. Due to the rise of the popularity of cryptocurrencies, EK authors have also shifted their focus to distribute cryptocurrency mining malware, which is classified as a new attack type called Drive-By-Mining (DBM) to deliver cryptojacking malware [19]. Another interesting finding in [18] was that EKs pose a significant threat, due to their intelligence to bypass contemporary security countermeasures. One such example is the fact that EKs have evolved beyond file-based methods, with malicious files now existing as memory processes in the operating system in order to evade detection (file-less payloads). This makes even more challenging the detection of EKs at the host level as the delivery payload does not touch the disk and leaves no room for antivirus technology to detect EKs [20]. Thus, despite their longevity, EKs are still relevant in the underground malware market as they evolve by adapting to new malware trends to be always up to date. The research community has been also active in this area (as it was analyzed in Chapter 4), while the latest works (i.e., published in 2020 & 2021) analyze EKs from various point of views including browser forensics, detection methodologies, and the role of underground economy in the continuous development of EKs [21], [22], [23]. However, there is a gap regarding detection of EKs via their HTTP network flows. This motivates the third research question:

RQ3*:* ***Can we build accurate EK detection systems that are capable of efficiently detecting state-of-the-art EKs based on traces at the network level operation of EKs?***

Fourth, several SSJI based security gaps on Node.js implementations have been brought to light. A notable case is a vulnerability discovered by a researcher regarding PayPal [2]. On top of that, already, a lot of SSJI vulnerabilities have been disclosed in Node.js modules, e.g. CVE-2014-72052. These discovered vulnerabilities affect numerous remote servers, thus severely impacting on multiple entities that rely on and interact with them. Moreover, SSJI attacks have a lot of gravity in terms of security impact, as it results in unauthorized access to remote servers. Indeed, compared to their client-side counterpart (i.e., XSS attacks), an SSJI can execute code in an unprotected environment and have access to the underlying system. On the other hand, XSS attacks are significantly limited by the browser's sandbox. Furthermore, the work in [3] observed that Node.js developers exhibit a reluctance to resolve security issues in their applications. This can be attributed to the fact that due to the scarcity of Node.js security analysis tools, the developers have not cultivated an information security mindset. Indeed, other code injection attacks such as SQL injection have been adequately studied by academia and there are plenty of researches in the literature [4] [5] [6]; however, without a doubt, the research on SSJI attacks is limited; consequently, there is a clear lack of methodologies and tools for the detection and mitigation of such threats. This issue will be further discussed in Chapter 5. Consequently, this leads to the fourth research question:

RQ4: ***Is it feasible to create a methodology to automatically detect and exploit SSJI vulnerabilities in Node.js applications?***

1.2 Research Contributions

Motivated by the above mentioned observations and based on the four RQs (RQ1-RQ4), this thesis led to four main research contributions which are described as follows:

[1] https://www.kaspersky.com/resource-center/definitions/what-is-maze-ransomware

C1: To address RQ1, this contribution elaborates on a domain-agnostic and large-scale survey of the defense methods designed to tackle AML attacks. The main objectives are to organize the existing knowledge and provide directions for future research to facilitate the scientific community so it can propose more efficient and robust defense solutions to address the ongoing threat of AML. More specifically five methodological steps have been followed. In the beginning, the current surveys in the AML field were studied, identifying their strengths and weaknesses and the proposed directions for future research. It should be noted that almost all the surveys in the field highlight that further research is needed to minimize the threat of AML. The core of this research is the investigation of the current defenses that focus on tackling AML in order to identify common methods and traits based on which a taxonomy will be created. To the best of our knowledge, this is the first attempt to introduce that introduces a domain-agnostic taxonomy of AML defenses (i.e., the defenses are belong to computer vision, cybersecurity, NLP, and audio domains) to systematize the existing knowledge and extract insights about the current state of the defenses by comparing them regarding several criteria, such as whether they are attack and/or domain-agnostic, their application domain, and their defense category. Later, the evaluation datasets and metrics of the literature were collected in an attempt to gather this information in one place to assist researchers in the evaluation of future AML defense solutions.

C2: To address RQ2, this contribution investigates a data-driven Ensemble Learning methodology that implements two different Ensemble Learning methods and highly informative hybrid features to efficiently detect phishing emails. The methodology aims to enhance the prediction results in the phishing email detection domain by classifying phishing emails accurately, quickly, and cost-effectively, exploiting all the information that emails' contain. Using hybrid features, namely features obtained both from the message body text (i.e., the email's message that is shown to the recipient) and the contents of emails (i.e., header fields, most used words in phishing emails, attachments, etc.), this thesis achieves to convert the email into a compact, yet thorough representation that can be efficiently interpreted by the Ensemble Learning methods to acquire high detection performance. The methodology is comprised of 6 stages. First, the *Email Parsing stage* divides the email contents and stores them in arrays to ease their processing. Next, the *Content-based Feature Extraction stage* extracts 22 features from the emails' contents, which are divided in four categories: body, syntactic, header, and URL. In parallel, the *Pre-processing stage* prepares the email's body text for the *Textual Feature Extraction stage,* which employs the Word2Vec method to acquire text-based features. Later, the *Feature Selection stage* deploys the Mutual Information method to select the most informative features of the hybrid feature sets. Last but not least, the binary classification of the emails between benign and phishing is performed in the *Ensemble Classification stage*, where two methods are proposed, Method 1 - Stacking Ensemble Learning and Method 2 - Soft Voting Ensemble Learning. These methods are comprised of two ML algorithms to process the hybrid features separately and in parallel (i.e., the Decision Tree ML algorithm processes the content-based features, while k-Nearest Neighbors process the text-based features).

C3: To address RQ3, this contribution introduces an ML-driven methodology, named EKnad (Exploit Kits' network activity detection) that can detect EK infection activity by analyzing the network traffic from packet captures. More specifically, EKnad contains two phases, the Pre-processing and the EK detection. The Pre-processing phase first extracts the HTTP communication from the network traffic. Second, it creates potential EK sessions by grouping the HTTP flows aiming to the better and faster identification of EKs redirection chains. Following, the EK detection phase extracts discriminating features from the potential EK sessions and feeds them to ML algorithms for EK classification between EK or benign. A comprehensive set of 47 different features has been proposed aiming to cover all the infection steps of EKs, which are divided into 5 categories: Header, Redirection, Evasion, URL & Domain, and Connection. To evaluate EKnad, publicly available EK network traces from 26 different EK families deployed to perform four rigorous and diverse evaluation experiments. Special care was taken so that the evaluation methodology complies with state-of-the-art guidelines that should be followed when evaluating ML algorithms to avoid misleading and erroneous results. Numerical results show that EKnad achieves the best detection performance with the Multilayer Perceptron (MLP) Artificial Neural Network classifier outperforming other ones (i.e., Random Forest, Bayesian Network, Decision Table, k-Nearest Neighbors, J48). EKnad is able to detect new and last-generation EKs based on the training of older EKs, as well as to detect less-known EK families based on the training of well-known EK families. In addition, EKnad outperforms famous rule-based intrusion detection systems including Snort and Suricata.

C4: To address RQ4, this thesis introduces the first methodology and its implementation as a software tool named NodeXP (NOde.js JavaScript injection vulnerability DEtection and eXPloitation) that automatically detects and exploits SSJI attacks. More specifically, first the software architecture and the approach that was followed for the design and implementation of NodeXP is analyzed. The detection process is done through dynamic analysis, using both result and blind -based injection techniques. Upon vulnerability detection, the exploitation process is initiated to establish a remote session with the server automatically. NodeXP provides several advanced functionalities that distinguish it from many of its peers. A novel aspect of NodeXP is that it obfuscates attack vectors - not to avoid reverse engineering - but to bypass application-level filtering mechanisms (e.g., blacklisting) as well as network-level mechanisms, like Web Application Firewalls (WAFs) and Intrusion Detection Systems (IDS). Next, a thorough assessment of the detection capabilities of NodeXP was performed on different testing scenarios to evaluate its efficacy and compare it with well-known commercial and open-source vulnerability scanners. Experimental results show that NodeXP outperforms state-of-the-art vulnerability scanners. Please note that the tool is released as open-source to drive more research in this area and facilitate the security community in testing and discovering SSJI vulnerabilities.

The contributions presented in this thesis have led to several peer-reviewed research papers that presented in Table 1.

Table 1 List of publications related with the contributions of this thesis

Title	Authors	Venue	Contribution
Defense Strategies for Adversarial Machine Learning: A Survey	P. Bountakas, A. Zarras, A. Lekidis, C. Xenakis	Computer Science Review (IF: 8.75) *Under Review*	C1
Helphed: Hybrid Ensemble Learning Phishing Email Detection [31]	P. Bountakas, C. Xenakis	Journal of Network and Computer Applications (IF: 7.57)	C2
A comparison of natural language processing and machine learning methods for phishing email detection [33]	P. Bountakas, K. Koutroumpouchos, C. Xenakis	Proceedings of the 16th International Conference on Availability, Reliability and Security (Ranking: B)	C2
EKnad: Exploit Kits' network activity detection [32]	P. Bountakas, C. Ntantogian, C. Xenakis	Future Generation Computer Systems (IF: 7.3)	C3
NodeXP: NOde. js server-side JavaScript injection vulnerability DEtection and eXPloitation [30]	C. Ntantogian, P. Bountakas, D. Antonaropoulos, C. Patsakis, C. Xenakis	Journal of Information Security and Applications (IF: 4.96)	C4

1.3 Thesis structure

The remainder of this thesis in organized as follows:

- Chapter 2: Studies existing defense methods against all types of AML attacks and elaborates on a taxonomy that classifies the defenses against various categories aiming to drive more research towards the detection and mitigation of AML attacks.
- Chapter 3: Investigates how ensemble learning methods when combined with hybrid features extracted both from the contents and body text of emails can tackle phishing email attacks. This chapter also contains a thorough assessment on a large imbalanced email dataset that consider the advancement of phishing emails.
- Chapter 4: Discusses an ML-driven methodology for the detection of EKs based on their network traffic and more specifically on the HTTP traces they leave behind during the infection as well as it presents the experimental evaluation that was based on several scenarios to highlight the efficacy of the methodology on different use cases.
- Chapter 5: Presents a novel platform for the detection and exploitation of SSJI vulnerabilities in Node.js applications along with a comparison with other well-known vulnerability scanners.
- Chapter 6: Summarizes the contribution of this thesis.
- Chapter 7: Concludes the thesis and highlights open problems for future research.

2. Survey: Defense Methods Against Adversarial Machine Learning Attacks

2.1 Introduction

The recent progress in Artificial Intelligence (AI) has led to its deployment into every technological domain, including but not limited to smart cities, self-driving cars, autonomous ships, 5G/6G, and next-generation intrusion detection systems. The utilization of AI almost in every aspect of our lives has introduced a new and enormous attack surface that targets Machine Learning (ML) and Deep Learning (DL) algorithms compromising their functionality and affecting their performance, acknowledged as Adversarial Machine Learning (AML). Considering the heavy applicability of ML and DL also in critical tasks (e.g., cancer diagnosis) [165], AML can lead to severe consequences, varying from data privacy leakage to even loss of life.

Motivated by the observations mentioned in Chapter 1, section 1.1 (i.e., limited research on the defense solutions against AML attacks, most research on AML field focus on computer vision, and AI systems developed based on the inaccurate assumption that the operation environment is trusted and secure), this chapter aims to answer the following research question:

Is it possible to create a taxonomy of existing defenses against AML attacks? (RQ1)

To answer RQ1 this chapter makes the following contribution:

Introduction of a domain-agnostic and large-scale survey of the defense methods designed to tackle AML attacks.

The objectives of this survey are to organize the existing knowledge and provide directions for future research to facilitate the scientific community so it can propose more efficient and robust defense solutions to address the ongoing threat of AML. Thus, the methodological steps depicted in Figure 2-1 have been followed. More specifically, in Step 1, the existing literature was collected utilizing a systematic methodology discussed in Section 2.2. In Step 2, the current surveys in AML field were studied, highlighting their strengths and weaknesses as well as their directions for future research, where almost all the surveys identified that further research is needed to minimize the threat of AML. Later, the existing defenses aiming to tackle AML were examined to identify common methods and traits based on which they will be categorized. In Step 3, a taxonomy of the existing defenses that aim to counter AML attacks was created. As it was mentioned in Chapter 1, section 1.2, this is the first attempt to introduce a domain-agnostic taxonomy of AML defenses to systematize the existing knowledge and extract insights about the current state of the defenses by comparing them regarding several criteria, such as whether they are attack and/or domain-agnostic, their application domain, and their defense category. The taxonomy is domain-agnostic since, instead of focusing on the defense methods of a particular domain, four different domains (cybersecurity, computer vision, NLP, and audio) have been considered. These particular domains have been selected because they are the most targeted domains of AML attacks [166]. In Step 4, the evaluation datasets and metrics were identified based on the existing literature in order to present the datasets and metrics that already exist and can be deployed for validating AML defense solutions. Finally, in Step 5, relying on the knowledge obtained from the taxonomy, numerous directions for future research were proposed.

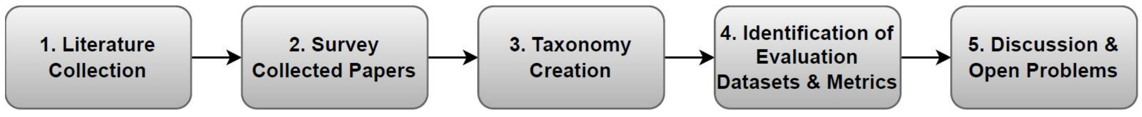

1. Literature Collection
2. Survey Collected Papers
3. Taxonomy Creation
4. Identification of Evaluation Datasets & Metrics
5. Discussion & Open Problems

Figure 2-1 Methodological Steps

The remaining of the Chapter unfolds as follows: Section 2.2 discusses the methodological steps that followed to collect the surveyed papers. Section 2.3 elaborates on background information related to

AML attacks. Section 2.4 presents the taxonomy of existing defenses against AML attacks. Section 2.5 focus on the existing datasets and evaluation metrics that could be used to evaluate defenses specifically created to tackle AML attacks, while section 2.6 discusses the findings of the research that was performed in this Chapter. Please note that parts of this Chapter have been submitted for publishing in *Computer Science Review*[2] journal and are currently under review.

2.2 Literature Collection

A systematic methodology was followed in order for this survey to detect and collect related research papers and surveys that tackle AML attacks. In particular, the following academic platforms: Google Scholar[3], ACM Digital Library[4], IEEE Xplore[5] ScienceDirect[6], SpringerLink[7], and arXiv[8] were scrutinized. In more detail, to collect the research papers for this study, the following queries were deployed: (i) adversarial machine learning defense, (ii) adversarial machine learning mitigation, (iii) adversarial training, and (iv) adversarial machine learning survey, as well as three selection criteria: (1) the subject of the paper aligns with the topic of the proposed survey; (2) the article was published after 2014; (3) the article is applied to one of the following domains: computer vision, cybersecurity, NLP, and audio.

Based on the criteria mentioned above, we concluded in the selection of 57 articles from which seven are existing surveys in AML field that mainly focus on the attacks and not the defenses, and 50 are proposing a defense method against one or more AML attacks in the fields of computer vision, cybersecurity, NLP, and audio classification. The selected papers were published between 2014 and 2022. More specifically, the allocation of the number of surveyed papers per year and domain is depicted in Tables 2 and 3, respectively. Please note that some defenses have been evaluated on several domains (e.g., image and audio classification), hence in Table 2, some papers are categorized in more than one domain.

Table 2 Allocation of the surveyed papers per year

Year	# Papers	Year	# Papers	Year	# Papers
2014	1	2017	6	2020	11
2015	1	2018	4	2021	14
2016	1	2019	7	2022	5

Table 3 Allocation of the surveyed papers per domain

Domain	# Papers
Computer Vision	16
Cybersecurity	17
NLP	6
Audio	13

2.3 Adversarial Machine Learning Attacks

This section presents an overview of ML/DL methods in cybersecurity domain and how such methods can be deployed to detect various cyberattacks (e.g., EK and phishing attacks). Nevertheless, the huge expansion of ML/DL methods has also introduced a new attack vector, where the ML/DL models can also be subject of cyberattacks.

The action of attacking AI systems (including ML/DL models) is known as Adversarial Machine Learning (AML). The main goal of AML is to influence the models' procedures and reduce their

[2] https://www.sciencedirect.com/journal/computer-science-review
[3] https://scholar.google.com
[4] https://dl.acm.org/
[5] https://ieeexplore.ieee.org/
[6] https://www.sciencedirect.com/
[7] https://link.springer.com/
[8] https://arxiv.org/

efficacy, for instance by adding slight perturbations to unseen data an adversary can affect the performance of pre-trained models. By introducing adversarial examples (perturbated data), AML attacks are able to deceive ML/DL detectors that are not trained to identify adversarial examples and increase the number of misclassifications, thereafter, bypassing the detector. AML attacks have been thoroughly studied in other domains, such as image classification and object detection. However, in the cybersecurity domain, AML has been received significant less attention, where existing research mostly focus on malware detection [100] and IDS [101], while AML in phishing and EK detectors are still at its infancy.

In the rest of the section the most well-known methods for the generation of adversarial examples are discussed along with their application in existing works that focus on attacking ML/DL detectors.

2.3.1 Adversary's Knowledge

An essential factor of an AML attack is the knowledge that the adversary has about the target model. Based on the amount of the adversary's knowledge and, as an extension, the access she has to the model, AML attacks can be classified into three categories: white-box, black-box, and gray-box.

White-box. In white-box attacks, the adversary has complete knowledge of the system, the algorithm's architecture, hyper-parameters, and training data. While the amount of knowledge that white-box attacks require is rarely available in real-life scenarios, it is not unrealistic to occur. This knowledge can be acquired either by reverse engineering the model or by an insider. For example, via reverse engineering, adversaries might recover the model's components (e.g., architecture, hyperparameters) and gain white-box access. As one can understand, white-box attacks are the most potent type of AML attacks; thus, they are usually deployed to evaluate the robustness of ML/DL models and defenses. Namely, if a model is robust against white-box attacks, it will also be robust against gray-box and black-box attacks.

Black-box. In a black-box attack, the adversary has no knowledge of the system or the deployed defenses. The only information the adversary has is that, given the input, she can monitor the outcome of the system. Nevertheless, in some cases, the number of queries she can perform is limited (e.g., it might arouse suspicion), which hardens her work. Szegedy et al. [102] was the first that observed the adversarial example transferability, namely adversarial examples created to be misclassified by a model are possible to be also misclassified by a different model, even when the models are trained on different datasets. Papernot et al. [15] carried out black-box attacks on a Deep Neural Network (DNN), based on the adversarial example transferability effect. The authors conducted the attacks by only observing the outputs of the DNN given particular inputs. Their attack strategy included a substitute model, which was trained using synthetic inputs labeled by the DNN model to craft adversarial examples that were used to attack the DNN model.

Gray-box. The gray-box attack refers to the attack paradigm, where the adversary has some degree of knowledge about the system or the deployed defenses. In real-life scenarios, commonly, the adversary cannot obtain white-box access; however, it may receive partial pieces of information, such as some of the classifier's features, the class label, or in the case of DNN, the results of the hidden layers.

2.3.2 Adversary's Capability

An adversary's capabilities can be grouped into two categories: attack influence and data manipulation. The attack influence can be further divided into causative attacks that happen when the adversary modifies the training data (e.g., in case a system is retrained) or exploratory attacks that occur when the adversary skews the data during the testing phase. On the other hand, data manipulation is the process that defines how the samples or features can be modified. For instance, it determines which features can be altered without compromising the scope of the original sample.

2.3.3 Attack Types

According to previous works [28] [29], there are two major types of attacks that an adversary can perform on ML/DL models: poisoning attacks and evasion attacks. Both attacks aim to maximize the generalization error of the decision-making process to affect the system's decision.

2.3.3.1 Poisoning Attacks

The poisoning attack is the attack type that targets the system's learning process during the training phase. The poisoning attacks can be further classified into three categories.

Training Data Modification was the first type of poising attack proposed in the literature. The modification of training data includes modifying or deleting training data or injecting adverarial examples on the training dataset [103].

Label Manipulation or Label Flipping occurs when the adversary alters the labels of the training samples. Hence, this method is only applicable to supervised learning tasks. Biggio et al. [104] presented that randomly flipping 40% of the training labels can significantly reduce the performance of SVM classifiers.

Input Feature Manipulation is similar to label manipulation attacks; however, instead of altering the labels, the adversary can manipulate the training samples' features parsed by the ML/DL algorithm [105].

2.3.3.2 Evasion Attacks

An evasion attack targets the testing phase of already trained ML/DL classifiers by manipulating the samples without having access to the training data aiming at violating the system's integrity [104]. Since these attacks are performed in the testing phase, they are the most practical attack types and are widely used. For instance, the injection of adversarial examples during the testing phase to affect the prediction performance of a classifier is considered an evasion attack. A typical evasion attack is an input/output black-box attack, where the adversary inserts arbitrary inputs on an ML/DL model to observe its output [106]. The attacker's objective hence violates the integrity of the system, either by conducting a targeted or an indiscriminate attack, depending on whether she is targeting a particular system or not.

2.3.4 Adversarial Samples Generation Methods

2.3.4.1 FGSM

Fast Gradient Sign Method (FGSM). Goodfellow et al. [107] introduced the FGSM method that computes the gradient of the cost function based on the model's input. More specifically, FGSM generates adversarial samples that are the original samples with a small perturbation (i.e., modification) to affect the prediction of the model (i.e., classifier). The adversarial samples are generated by the following equation:

$$X^{*} = X + \varepsilon sign\left(\nabla_{x} J(\theta, X, y)\right),$$

where $X*$ is the adversarial sample, ε indicates the value that controls the magnitude of the perturbation (i.e., the noise that will be applied), J represents the cost function of the model, ∇x is the gradient of the model based on the sample X and its label y, and θ is the model's hyper-parameters.

FGSM, except for its application in the image classification field, has also been employed in various domains, for instance, cybersecurity [108]. However, as it was mentioned by Rosenberg et al. [9], the application of FGSM in the cybersecurity domain is more challenging than in computer vision due to the nature of the samples.

2.3.4.2 C&W

Carlini & Wagner (C&W)[9] introduced an effective adversarial sample generation method that identifies the less possible perturbations based on a distance metric (L_0, L_2, L_∞). The L_0 and L_2 concluded to have much fewer perturbations than SOTA (FGSM and JSMA). The aim of C&W is to minimize $D(x, x + \delta)$ such that $C(x + \delta) = t, x + \delta \in [0, 1]^n$, where (x) is the sample, δ is the perturbation, Δ is the distance metric. Namely, the goal is to find a small alteration δ to make on the sample that will change its classification without ruining its type (e.g., if the sample is an image and a perturbation δ is added, then the result should be still a valid image).

2.3.4.3 JSMA

Jacobian based Saliency Map Attack (JSMA). Papernot et al. [109] presented the JSMA method that produces adversarial samples using saliency maps and employs the Jacobian matrix of the model to calculate the sensitivity direction. The Jacobian matrix of a model F with respect to the input samples X is calculated via the equation below:

$$J_F = \frac{\delta F(X)}{\delta X}$$

Later, J_F is deployed to generate adversarial saliency maps used to identify the features of the input data most relevant to the model's decision. Namely, the features that should be perturbed to affect the model's classification. Finally, the JSMA method checks whether the alterations have affected the model to misclassify the input. If they have not affected it, another set of features is selected, and the process repeats until an adversarial saliency map arises that can be utilized to generate an adversarial sample.

2.3.4.4 DeepFool

Moosavi et al. [110] proposed a repetitive procedure for creating adversarial samples. This method minimizes the Euclidean distance between the adversarial and original samples. DeepFool is comprised of the calculation of a linear decision limit, which divides the samples that belong to different classes. In DNN is difficult to find linear decision limits, hence the perturbations are added repetitively by applying this method multiple times. The robustness of a classifier is calculated by the next equation:

$$\Delta(x, F) = min_r ||r||_2, F(x + r) \neq F(x),$$

where $\Delta(x, F)$ is the robustness of classifier F given a sample x and r is the minimal perturbation that it can alter the estimated label $F(x)$. DeepFool is a powerful method for generating adversarial samples with fewer perturbations than FGSM and JSMA; however, it is computationally expensive.

2.3.4.5 BIM

Kurakin et al. [111] proposed an adversarial sample generation method based on FGSM. More specifically, BIM is an extension of FGSM, meaning that it adopts an iteration process to perturbative the pixels of an image, ensuring that the values are pruned within a certain margin of the source image. Even though BIM has been tested in the image classification domain, it has also been applied in the IoT domain and in intrusion detection.

2.3.4.6 L-BFGS

Szegedy et al. [102] was one of the first who observed that adding perturbations to images can deceive DL models and lead to misclassifications. Notably, the authors generate adversarial samples by adding perturbations to an image having as a scope to reduce the added perturbation r under L_2 distance:

[9] Carlini, N., & Wagner, D. (2017, May). Towards evaluating the robustness of neural networks. In 2017 ieee symposium on security and privacy (sp) (pp. 39-57). Ieee.

$$min||r||^2, f(x+r) = y,$$

where r is the perturbation, f is the loss function of the model, x is the original image, and y is the incorrect prediction label. Because the above equation is non-trivial, the authors deployed a box-constrained L-BFGS to solve it. Even though the L-BFGS method is efficient at generating adversarial samples, it is computationally ineffective.

2.3.4.7 Adversarial Machine Learning Attacks Application to Intrusion and Malware Scenarios

Rigaki, was one of the first that evaluated AML attacks on ML-based IDSs. More specifically, the authors employed the NSL-KDD dataset and FGSM and JSMA methods to generate the adversarial examples. The evaluation performed on five models, Decision Tree, Random Forest, SVM, Multilayer Perceptron, and voting ensemble learning. The authors made an important remark regarding the percentage of features that each attacks modifies, namely FGSM modifies 100% of each sample's features, while JSMA modifies only 6%. Furthermore, all the classifiers were affected significantly by the AML attacks, however, Random Forest was the most robust classifier with 18% accuracy and 6% F1-score and AUC drop.

Towards the same direction Wang et al.[10] employed various AML adversarial example generation methods, such as FGSM, JSMA, Deepfool, and C&W to evaluate DL-based IDS. The authors deployed the NSL-KDD dataset and a Multilayer Perceptron model, and the results concluded that the C&W was the least effective attack, while the FGSM was the most effective one. In this work, the authors also discussed a manual attack where the adversary manually modifies the features to perform an adversarial attack. The same AML attacks also deployed by Martins et al.[11], where the authors deployed the NSL-KDD and the CICIDS2017 datasets and multiple classifiers (e.g., SVM, decision tree, random forest, naïve bayes, neural network and denoising autoencoder). AUC decreased by 13% on NSL-KDD dataset and 40% on the CICIDS, which was attributed to the fact that CICIDS dataset contains more features, hence it is more vulnerable.

Xu et al. proposed MANIS [112], a system that can evade graph-based malware detection systems. The authors deployed two attacking approaches: the n-strongest nodes and FGSM. Furthermore, MANIS takes as input the whole graph via adversarial example elaboration. The Drebin dataset was used as well as both white-box and gray-box adversarial settings. In white-box settings using the n-strongest node attack MANIS concluded to 72.2% misclassifications with 22.7% of the nodes injected, while using the FGSM attack, it was attained 33.4% misclassifications with 36.34% injection. The authors mentioned that in gray-box settings the performance reduction is equally significant.

Moreover, several other works have employed manual approaches to evaluate the robustness of ML/DL systems by manually interfering the datasets' features or utilized Generative Adversarial Networks to perform AML attack[12]. Lin et al. [101] deployed the GAN method to generate adversarial examples that were used to attack various classifiers (e.g., SVM, naïve Bayers, MLP, etc.).

[10] Wang, Z. (2018). Deep learning-based intrusion detection with adversaries. *IEEE Access*, *6*, 38367-38384.
[11] Martins, N., Cruz, J. M., Cruz, T., & Abreu, P. H. (2019). Analyzing the footprint of classifiers in adversarial denial of service contexts. In *Progress in Artificial Intelligence: 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Vila Real, Portugal, September 3–6, 2019, Proceedings, Part II 19* (pp. 256-267). Springer International Publishing.
[12] Apruzzese, G., Colajanni, M., Ferretti, L., & Marchetti, M. (2019, May). Addressing adversarial attacks against security systems based on machine learning. In *2019 11th international conference on cyber conflict (CyCon)* (Vol. 900, pp. 1-18). IEEE

A similar approach followed by. Feng et al.[13], where they also deployed GAN to generate adversarial examples and target an LSTM neural network.

On the other hand, the majority of defenses against AML attacks in the cybersecurity domain are focusing on the adversarial training method. In adversarial training, the main goal is to improve the ML/DL model's resilience against AML attacks by deploying in the training phase both adversarial and benign examples. A training dataset D that is comprised by the tuple X = (x, y) (x is the sample and y is its label), an adversarial example $x*$ is generated by a method M (e.g., FGSM) based on x and results in a new tuple $X*$ with the same label (y), namely $X* = (x*, y)$. Later, D will include also X∗ resulting in $T = \{X, X*\}$. The ML/DL algorithm is then retrained using the new dataset, which will conclude to a more robust model.

Some popular existing works that employ adversarial training are discussed in the following. Khamis et al. [108] was one of the first that brought adversarial training in the cybersecurity domain. Particularly, their approach was based on the combination of min-max formulation with adversarial training. The experimental results on various algorithms and AML attacks showed that the evasion rate was reduced significant. The authors extended their previous work and implement min-max adversarial training in three different deep neural networks (ANN, CNN, RNN), where they concluded that the adversarial training can be considered a reliable defense method. Along the same the authors of [113] also used min-max adversarial training in a Multilayer Perceptron (MLP) classifier that was evaluated using three AML attacks, named FGSM, PGD-K L2, and PGD-K L∞. The results indicated that PGD with a larger K value (e.g., PGD-40 min-max training) provides better robustness.

Apruzzee et al. [114] applied adversarial training in the field of botnet detection introducing a Deep Reinforcement Learning method that generates adversarial examples automatically. The adversarial examples were created by changing some essential traffic flow features (e.g., duration, sent bytes, received bytes). The evaluation resulted in an improved detection rate from 0.58 to 0.88 without sacrificing the performance on non-adversarial examples. Towards the same direction, Anthi et al. [115] deploy adversarial training to enhance the robustness of tree-based classifiers (Decision Tree and Random Forest). The adversarial examples were crafted employing the JSMA method, and the detection performance of Random Forest and Decision Tree models decreased by 6% and 11%, respectively. However, using adversarial training and only 10% of the adversarial examples to retrain the classifiers, Random Forest and Decision Tree models accomplished 11% and 17% higher F1-scores respectively.

Debicha et al. [116] implement adversarial training in a DL-based IDSs by using the FGSM, BIM, and PGD methods to generate adversarial examples. Adversarial training based on PGD adversarial examples managed to increase the model's robustness significantly; nevertheless, the authors highlighted that there is a trade-off of slightly decreasing the detection accuracy on clean data, namely when there are no adversarial examples. An important observation the authors made was that increasing the percentage of adversarial examples in the training phase does not necessarily enhance the model's robustness. Rashid et al. [117] studied the effects of AML attacks on ML/DL-based IoT IDSs and introduced an adversarial training method to improve the models' robustness. The authors employed three classifiers (MLP, DT, and RF) and evaluated the five most known AML attacks (JSMA, FGSM, BIM, DeepFool, and C&W). The authors concluded that without adversarial training the MLP had relatively higher robustness than the other ML classifiers. For the adversarial training, the authors deployed the adversarial examples that were created using the C&W method, where 60% of the samples were added to the training set, and the remaining 40% was included in the testing set.

[13] Feng, C., Li, T., Zhu, Z., & Chana, D. (2017). A deep learning-based framework for conducting stealthy attacks in industrial control systems. *arXiv preprint arXiv:1709.06397*.

The results showed that adversarial training is a sufficient defense method against adversarial AML attacks.

2.4 Taxonomy of Adversarial Machine Learning Defenses

The rise of AML attacks has inspired the scientific community to drive more research on designing and developing novel defense methods against such attacks. This section elaborates on a taxonomy of AML defense methods, which to the best of our knowledge, is the first large-scale taxonomy of the available defenses against AML attacks. As it is shown in Figure 2-2, the taxonomy includes two main axes based on the defense objective: (i) Mitigation and (ii) Detection. Finally, the taxonomy analyzes various types of AML attacks on four pioneer domains that gather the most works of the literature (i.e., cybersecurity, computer vision, NLP, and audio).

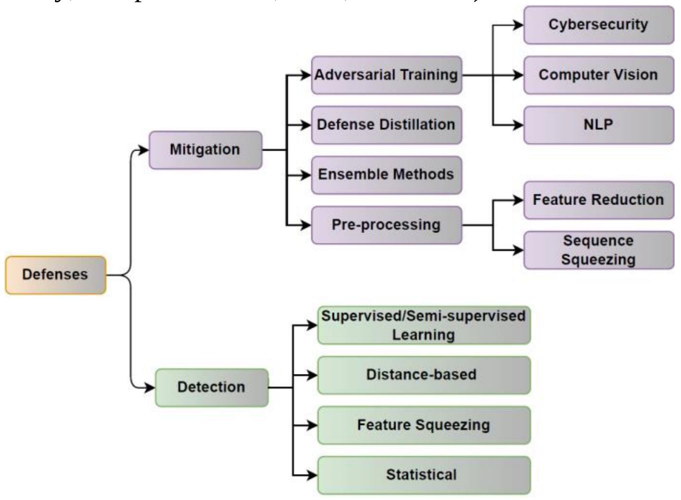


Figure 2-2 Taxonomy of AML defense methods

2.4.1 Mitigation

The mitigation category refers to proactive defenses that strive to enhance the robustness of ML/DL models against AML attacks.

2.4.1.1 Adversarial Training

The adversarial training defense method aims to enhance the model's robustness by utilizing a training dataset that contains both benign and adversarial examples. The difference between traditional and is that in adversarial training both original and perturbated data are deployed in the learning phase. Specifically, given a training dataset D that contains the tuple $X = (x, y)$ (where x is the sample and y is the label), an adversarial sample x∗ is crafted by an algorithm A based on x and forms a new tuple $X*$ that has the same label (y) as X, namely $X* = (x*, y)$. Later, the training dataset D will be expanded with $X*$ resulting in $T = \{X, X*\}$. The ML/DL algorithm is then retrained using the expanded dataset, which will result in a more robust model in theory.

Adversarial training is the literature's most well-known defense method against AML attacks. To this end, most of the existing defenses are based on different variations of adversarial training. It should be noted that because the majority of defenses are based on adversarial training, this section categorizes the defenses of this category based on their application domain.

Computer Vision

Adversarial training was first introduced in the computer vision domain [107]; hence it has received considerable attention from the research community during these years. Goodfellow et al. [107] were among the first that raised the adversarial training defense in the computer vision field. The authors deploy the FGSM method to generate adversarial examples and, via adversarial training, managed to decrease the error rate of a DNN image classifier from 89.4% to 17.9%. The experiments were performed on the MNIST dataset[14].

Huang et al. [167] studied the role of adversarial training on classifiers' robustness and proposed a new approach to generate adversarial examples. The authors employed a min-max adversarial training approach, i.e., Learning With an Adversary (LWA), to train a DNN image classifier. The experimental results indicate that LWA accomplishes better robustness and high accuracy on original samples compared to existing methods (e.g., [107]) on both MNIST and CIFAR-10[15] datasets.

Madry et al. [168] deployed PGD and FGSM methods to generate adversarial examples, which were used to retrain a DNN. The authors deployed black-box and white-box adversary's knowledge and proved that the proposed adversarial training approach could achieve robustness against different attacks. Notably, the DNN trained with the MNIST dataset accomplished 89% accuracy on the strongest AML attack (white-box & PGD), while on the same attack, the DNN trained with CIFAR-10 dataset achieved 46%. The drawback of the proposed defense is that it is computationally complex; thus, it is difficult to test it on large datasets.

Sankaranarayanan et al. [169] proposed a new regularization approach for DNN image classifiers that improves the adversarial robustness in comparison with traditional adversarial training methods. More specifically, the authors compared the proposed layer-wise adversarial training with FGSM and FGSM-inter (i.e., a variation of FGSM where intermediate layer activations are perturbed). They concluded that when perturbating intermediate layers and not only the input layer of a DNN, the classification performance is improved. However, when perturbating only the input layer the adversarial robustness is improved. Thus, there is a trade-off between the adversarial robustness and the regularization effect on original data.

The authors of [170] proposed an adversarial training algorithm that eliminates the additional time that emerges from the generation of adversarial examples. The algorithm eliminates this overhead by recycling the gradient information computed when updating the model's parameters. The proposed algorithm needs almost the same training time as in original training to produce robust DNN models, and it is 3 – 30 times faster than previous works (e.g., [168]). The experimental results concluded that the produced models accomplish slightly better accuracy on CIFAR-10 and CIFAR-100 datasets than conventional adversarial training methods. Furthermore, the proposed algorithm can also be applied on large-scale datasets, like ImageNet attaining 40% accuracy against PGD attacks.

Xie et al. [171] combined feature denoising with adversarial training to enhance CNNs' adversarial robustness. The proposed method improves the adversarial robustness in white-box and black-box adversarials' knowledge against PGD AML attacks. More specifically, the authors deployed the large ImageNet dataset, and in a 10-iteration PGD white-box attack, the proposed method achieved 55.7% accuracy (the prior state-of-the-art has 27.9% accuracy). Moreover, in the extreme case of 2000-iterations PGD white-box attacks, the proposed method attained 42.6% accuracy, while in a black-box setting against 48 unknown attackers, the proposed method achieved 50.6% accuracy, and it was ranked first in Competition on Adversarial Attacks and Defenses 2018.

The authors of [172] introduced the concept of combining ensemble learning with adversarial training. Their approach merges the training data with adversarial perturbations generated from three different AML attacks instead of one. Specifically, the authors deployed the FGSM, the Single-step Least-Likely Class Method (Step-LL), and the iterative I-FGSM. The experiments were performed using several variations of Convolutional Neural Network (CNN) and Fully Connected Network (FCN) models on the ImageNet dataset with white-box and black-box adversary's knowledge. The

[14] http://yann.lecun.com/exdb/mnist

[15] https://www.cs.toronto.edu/ kriz/cifar.html

results indicate that ensemble adversarial training is not robust against white-box I-FGSM and Step-LL, while it enhances the models' robustness against black-box attacks.

Lal et al. [173] proposed a robust image classification framework for detecting Diabetic Retinopathy. The authors employed a CNN model (Darknet53) to perform the classification and deployed an adversarial training approach to enhance the model's robustness against AML attacks. The FGSM, DeepFool, and Speckle Noise AML attacks have been used to evaluate their framework. Four different adversarial training methods were employed, and the best robustness was accomplished by mixed adversarial training (MAT) in which the training dataset included adversarial examples from all the AML attacks. To be more precise, MAT achieved 92% correct label prediction.

Ding et al. [174] developed a method for restoring images from degradation focusing on visual context prediction in the case of autonomous vehicles. The proposed method is based on a multi-stage conditional Generative Adversarial Network (cGAN) for image restoration that employs adversarial training to detect images from various types of degradation. The authors performed the adversarial training using both original and degraded images using various degradation methods, such as motion blurring, Gaussian filtering, average filtering, Gaussian noising, oversharpening, and JPEG compression. The evaluation results concluded that the proposed method is more efficient for image restoration than the existing works and other methods (CycleGAN, DeblurGAN).

Xu et al. [175] proposed a Fair-Robust-Learning (FRL) framework to mitigate the disparity of accuracy and robustness between different classes introduced by adversarial training. The authors increased the perturbation margin ε during training in the generation of adversarial examples process for a specific class. This way, the class's robustness is improved, and the boundary error is reduced. The FRL framework employed a DNN model and was evaluated using the CIFAR-10 and SVHN[16] datasets. Moreover, FRL improved the lower robust error of [33] and [41] by 10% – 15%.

CyberSecurity

Khamis et al. [108] proposed one of the first adversarial training approaches in the cybersecurity domain. Their approach was based on the min-max formulation aiming to enhance the robustness of ML/DL-based IDSs. The authors developed several ML and DL classifiers, such as DNN, RF, AdaBoost (AB), Naive Bayes (NB), and Support Vector Machine (SVM), which were trained on the UNSW-NB15[17] dataset. Four AML attacks were investigated (dFGSM, rFGSM, BCA, and BGA) in white-box adversary's knowledge that was performed in the DNN. In the baseline model, the dFGSM attack reached the higher evasion rate (100%), rFGSM and BGA yielded 99%, while BCA attained 64.4%. Using min-max adversarial training, the experimental results show that the evasion rate was significantly reduced. Moreover, the authors also concluded that the PCA feature selection method enhances the model's robustness against AML attacks.

An extension of the previous work was proposed in [113], where the authors deployed three different DNN DL-based IDS on Artificial Neural Network (ANN), CNN, and Recurrent Neural Network (RNN). The IDSes were trained to detect malicious network traffic on two well-known datasets (UNSW-NB15 and NSL-KDD[18]). Using white-box adversary's knowledge, the authors performed five AML attacks (FGSM, BIM, PGD, C&W, and DeepFool). The AML attacks were highly effective since the accuracy of the IDSes decreased from approximately 20% to 90% (C&W AML attack against RNN on the NSL-KDD dataset). However, in many cases, the adversarial training-based min-max formulation managed to reach the baseline results of the DL-based IDSes showing that it could

[16] http://ufldl.stanford.edu/housenumbers/

[17] https://research.unsw.edu.au/projects/unsw-nb15-dataset

[18] http://www.di.uniba.it/ andresini/datasets.html

be considered a reliable defense method. Furthermore, the authors highlight that C&W and DeepFool AML attacks on RNN are the most difficult to tackle.

Fu et al. [176] studied the robustness of three DL-based IDS against AML attacks. Particularly, the authors utilized the FGSM method to generate adversarial examples to evaluate the robustness of the CNN, LSTM, and GRU models. The models were trained with three different methods (normal training, adversarial training, and adversarial re-training). The experiments were performed on the CSE-CIC-IDS 2018 dataset. After the FGSM attack, the models' accuracy decreased by 20-40%, while the least affected model was the CNN. After the adversarial training and adversarial re-training, the models' robustness increased significantly. However, a drawback of this work is that the performance against the original samples was decreased compared to the results when trained only with the original samples. Another work along the same line is proposed in [44], where the authors also deployed min-max adversarial training. In this work, only a Multilayer Perceptron (MLP) classifier was evaluated using three AML attacks, named FGSM, PGD-K L2, and PGD-K L∞, without discussing the adversary's knowledge. The authors concluded that PGD with a larger K value (e.g., PGD-40 min-max training) offers increased robustness.

Apruzzee et al. [114] presented a Deep Reinforcement Learning to automatically generate adversarial examples in the form of network flows against botnet detectors. The adversarial examples were generated by altering a few essential traffic flow features (e.g., duration, sent bytes, received bytes, and transmitted packets). The generated adversarial examples were deployed to create an augmented dataset that, via adversarial training, can enhance the resilience of botnet detectors. The authors utilized two datasets in their experiments, CTU and Botnet, and two ML botnet detectors based on RF and Wide and Deep Learning (WnD). The evaluation results concluded that the hardened botnet detectors increased the detection rate from 0.58 to 0.88 against the adversarial examples without affecting the performance when tested with non-adversarial examples.

Towards the same direction, Anthi et al. [115] utilized adversarial training to improve the robustness of two classifiers (J48 decision tree and RF) designed to detect attacks on ICS. The authors performed a data poisoning attack based on the gray-box adversary's knowledge; they only had access to the dataset and the features, while they had not any other knowledge about the model. The adversarial examples that were utilized to perform the AML attack were generated using the JSMA method, and the detection performance of RF and J48 models decreased by 6% and 11%, respectively. As a countermeasure to this data poisoning attack, the authors employed the adversarial training method using 10% of the adversarial examples to retrain the classifiers. Afterward, the RF and J48 models were applied to the unknown adversarial examples, where they accomplished 11% and 17% higher F1-scores respectively.

Vitorino et al. [177] introduced the Adaptive Perturbation Pattern Method (A2PM) to generate realistic adversarial examples. The authors showcased their method on two scenarios (i.e., Enterprise and IoT in a gray-box attack setting) using the MLP and RF models, where they only had knowledge about the features. The adversarial examples significantly lowered the performance of MLP and RF in both scenarios. Furthermore, the authors evaluated the models also using adversarial training and concluded that when the models are adversarially trained their robustness against adversarial examples is significantly improved.

Debicha et al. [116] studied the effect of adversarial training in a DL-based IDS. More specifically, the authors performed white-box AML attacks using the FGSM, BIM, and PGD methods. The experiments were performed on the NSL-KDD dataset, where the baseline model reached 99.61% detection accuracy. The FGSM attack decreased the accuracy to 14.13%, while BIM and PGD reduced it even more to 8.85%. After the adversarial training using the PGD adversarial examples, the model's robustness increased significantly (it is not mentioned exactly how much); nevertheless, the authors highlighted that there is a trade-off of slightly decreasing the detection accuracy on clean data, namely when there are no adversarial examples. In addition, the authors concluded that increasing the percentage of adversarial examples in the training phase does not necessarily enhance the model's robustness.

Rashid et al. [117] investigated the effects of AML attacks on ML/DL-based IDSs designed to detect attacks against IoT smart city applications and proposed a defense method based on adversarial training to enhance the models' robustness. The authors utilized three classifiers (MLP, DT, and RF) and trained them using the DS2OS dataset[19]. Moreover, they evaluated the five most known AML attacks (JSMA, FGSM, BIM, DeepFool, and C&W); however, it is not mentioned what the adversary's knowledge is (e.g., white-box, black-box, etc.). All the AML attacks significantly decreased the performance of the models varying from 36% to 78%; however, the JSMA and C&W attacks were the most successful. The authors concluded that the MLP shows relatively higher robustness than the other shallow ML classifiers. For the adversarial training, the authors deployed the adversarial examples that were generated by the C&W method, where 60% of the samples were added to the training set, and the remaining 40% was included in the testing set. The authors concluded that the adversarial training is sufficient to maintain the classifier's performance against adversarial AML attacks. The baseline classifiers MLP, DT, and RF accomplished 99.43%, 99.35%, and 99.41% accuracy, while the re-trained classifiers achieved 99.20%, 99.19%, and 99.19%, respectively.

Natural Language Processing

In the NLP domain, there are several works in the literature, such as [178] [179] [180] [181] that deploy adversarial training as a mean to improve the prediction performance of NLP models rather than enhancing their robustness against AML attacks. Furthermore, other works that aim to enhance the NLP model's robustness are usually focused on BERT NLP models [182]. In the following, we will focus on the adversarial training methods that aim to mitigate AML attacks in the NLP domain.

Liu et al. [183] presented a comprehensive study on adversarial pre-training aiming to enhance both the generalization and robustness of NLP models even though prior works treat them as inversely related terms. The authors proposed an algorithm, which combines the training process with applying perturbations in the embedding space. The algorithms were applied to BERT and RoBERTa NLP models and tested against various NLP tasks using three adversarial NLP benchmarks and concluded that they could improve both the generalization and robustness of BERT and RoBERTa models.

Yoo et al. [184] argued that in NLP the generation of adversarial examples is a combinatorial optimization problem, which is solved by deploying a heuristic algorithm. To this end, the authors proposed two methods to generate adversarial examples and improve the adversarial training process: (i) a cheaper gradient-based word importance ranking to sequentially replace each word with a synonym (A2T) and (ii) a masked language model-based word replacement (A2T-MLM). The proposed methods tested both BERT and RoBERTa NLP models on text classification tasks using well-known datasets (e.g., IMDB, Rotten Tomatoes, and SNLI). The experimental results showed that the A2T method could improve the NLP model's robustness against several attacks of the literature that target NLP models.

Pan et al. [185] proposed an adversarial training method for text classification models. The authors based their method on white-box adversary's knowledge. The adversarial sample generation process was based on FGSM and [178], but instead of applying the perturbations to the word embeddings, they applied them to the word embedding matrix of Transformer encoders. The experiments were performed using the BERT and RoBERTa NLP models on three datasets and compared on several tasks with other adversarial learning methods (e.g., PGD, FreeAT, FreeLB) [181]. The results showed

[19] https://www.kaggle.com/francoisxa/ds2ostraffictraces

that the proposed method outperforms PGD and FreeAT, while accomplishing the same performance as FreeLB.

Kitada et al. [179] introduced an adversarial training method that can be applied to various NLP tasks. The proposed adversarial training method is focused on the attention mechanism. More specifically, the generated adversarial perturbations are added to the attention mechanism to increase the model's prediction performance and robustness. To test their method, the authors deployed a Bidirectional RNN (BiRNN) model and three different NLP tasks, text classification, question answering (QA), and natural language inference (NLi). The evaluation experiments concluded that the proposed adversarial training on the attention mechanism managed to improve the model's performance on the three NLP tasks and outperformed other training techniques.

Finally, Morandi and Samwald [186] performed a large-scale investigation of the robustness of the biomedical NLP models against AML attacks. The authors performed white-box and black-box attack scenarios on both character-level and word-level AML attacks. More specifically, five biomedical/clinical textual datasets were used and four different NLP models were deployed based on BERT. Moreover, four AML attacks were performed and the results indicated that the NLP models are not robust to AML attacks. The authors employed adversarial training using adversarial examples generated by all the tested attacks and evaluated the performance of the models against adversarial and clean test samples to measure the robustness and generalization, respectively. The proposed approach tested several NLP tasks, such as question answering, semantic similarity estimation, natural language inference, text classification, and relation classification. The experimental results demonstrated that adversarial training improves the robustness and the generalization ability of biomedical NLP models.

Audio

In the audio domain, adversarial training has not received much attention, since it has been employed only in a few existing works.

Pal et al. [187] proposed a hybrid adversarial training method to improve the robustness of speaker recognition systems. For this purpose, the authors provide diverse and robust adversarial perturbations for adversarial training as a defense for deep speaker recognition systems against various types of attacks. Experiments were conducted on various white-box and black-box attacks, such as the FGSM, the C&W, the PGD, and the Feature Scattering attack (FS) [188]. The proposed hybrid adversarial training approach outperformed other adversarial training methods (e.g., using only FGSM, PGD, or FS adversarial examples) on all the testing AML attacks as well as on the detection of original examples.

Other defense methods against audio AML attacks relied on adversarial training in audio signals using a denoiser model [189]. The proposed defense is based on the adversarial training proposed in [168]. However, instead of adversarially training a model from scratch that leads to convergence issues in Automatic Speech Recognition (ASR), the authors present a method to fine-tune a pre-trained ASR model using PGD adversarial examples. Then, the denoiser, which is pre-trained using deep regression, maps adversarial signals to benign ones. Experiments have been conducted using the FGSM and the PGD methods and demonstrated promising results. Nevertheless, more extensive tests still have to be conducted.

2.4.1.2 Defensive Distillation

The term distillation originally refers to the training procedure where a DNN model (student) is trained using knowledge transferred from a different DNN (teacher) [190]. The motivation for the use of distillation stems from the deployment of DL models in smaller devices (e.g., smartphones) that do not have the resources to perform complex computations, hence via distillation during the training phase, the knowledge is transferred from more complex DNN models.

Papernot et al. [191] employed an alternative to the distillation method to protect DNN against adversarial examples in the computer vision domain. They proposed the defensive distillation where instead of transferring knowledge between different DNN models, they extracted knowledge from a DNN to enhance its robustness against adversarial examples. The authors deployed the JSMA method to craft adversarial examples and two datasets (MNIST and CIFAR-10) to evaluate the defensive distillation. The defensive distillation method reduced the success rate of the adversarial examples from 95.89% to 0.45% on the MNIST dataset and from 87.89% to 5.11% on the CIFAR-10 dataset. Another work that studies the distillation method in AML domain is proposed in [192]. The authors presented the Adversarially Robust Distillation (ARD) method that combines distillation with adversarial training to transfer knowledge from robust teacher against AML attacks DNN models to student models. The experiments were performed using the adversarial training method proposed in [168] to train DNN models both with and without the ARD method on CIFAR-10 and CIFAR-100 datasets. The evaluation concluded that the ARD-trained models outperform the adversarially trained models.

Soll et al. [193] applied the defensive distillation method of the image classification domain to text classification. Particularly, the authors generated the text adversarial examples based on [194] and deployed two datasets (AG's corpus of news articles and amazon movie reviews) to test the defensive distillation on a CNN model. The experimental results indicated that the defensive distillation method did not have the same effect in the text classification domain as in image classification since the robustness of the CNN against the adversarial examples slightly increased.

Apruzzese et al. [195] proposed a defensive distillation approach for enhancing the robustness of RF in the cybersecurity domain (botnet network traffic detection). More specifically, the authors employed two RF models, one model to generate probability labels, namely they used the result of each Decision Tree (DT) of the RF ensemble algorithm to generate the probability for each sample to be malicious or benign, and then a second model was trained based on these probabilities. The proposed method evaluated adversarial examples only in the testing phase (i.e., the authors performed an evasion attack) generated by altering specific features of the dataset. The results concluded that it outperformed an undistilled model on both adversarial and non-adversarial examples. Moreover, the authors compared their method with other defense methods against AML attacks (e.g., adversarial training and feature reduction) and concluded that the proposed defensive distillation provides more resilience to RF against AML attacks.

2.4.1.3 Ensemble Methods

The Ensemble Methods category is based on the concept of Ensemble Learning, namely multiple ML algorithms are combined to solve a learning problem (e.g., classification). Existing works in this category employ Ensemble Learning to enhance the resilience of ML and DL models against AML attacks.

Appruzzese et al. [196] proposed an approach named AppCon to enhance the resilience of ML (DT, RF, and AB) and DL (MLP and WnD) network IDS. AppCon is based on the paradigm of Ensemble Learning, namely it employs an ensemble of detectors, where each detector is devoted to a particular web-application (e.g., Skype, WhatsApp. etc.) and the results of the detectors are used to train and test a hardened detector. The authors performed a gray-box adversarial attack adding small perturbations on specific features of the original botnet detection dataset (CTU-13). The experimental results indicated that AppCon could prevent 75% of adversarial examples without affecting the performance in non-adversarial settings.

Jiang et al. [197] proposed a framework, named FGMD, which can effectively detect malicious IoT network traffic while it is also robust against AML evasion attacks. The authors proposed an adversarial sample generation approach that considers the constraints that arise due to the inherent properties of the network traffic (e.g., it cannot modify all the features of the network traffic, the attack function of the original malicious samples must be preserved, etc.). To this end, the authors altered only four features (from 23) to generate the adversarial examples and perform an evasion attack with black-box adversary knowledge. The FGMD framework is based on feature grouping and ensemble learning (referred to as model fusion). More specifically, the authors grouped the features into three categories and parsed them by three different LSTM models to perform the detection. The final prediction of the FGMD is calculated based on the average of the classification of the three models. The evaluation experiments were performed on two IoT datasets (MedBIoT and IoTID) and FGMD compared with baseline models (LSTM, RNN, DT), both original and adversarially trained. The results indicate that FGMD can achieve higher scores (accuracy, F1-score, precision, recall, FPR) than other models in the presence of AML evasion attack.

Ensemble methods are also used for the detection of adversarial audio samples. In [198], the authors investigate the impact of Gaussian smoothing on the DeepSpeech and Lingvo ASRs. Specifically, the authors present four defense systems, namely randomized smoothing, Defense GAN (uses the same technique as with images [199]), Variational autoencoder (VAE), and Parallel WaveGAN vocoder (PWG). The systems were compared with the standard defense systems for adversarial training. As an outcome, the authors propose a hybrid system where PWG is combined with randomized smoothing to provide 93% accuracy and 90% detection improvement for C&W and BIM attacks, which impose the greatest highest risk against audio systems.

2.4.1.4 Pre-processing

The Pre-processing category is comprised of existing works that utilize a pre-processing technique (e.g., feature reduction) to improve the model's robustness against AML attacks.

Rosenberg et al. [200] proposed an attack-agnostic defense method for RNN classifiers against AML attacks that are named *Sequence Squeezing* based on an existing work proposed in [201]. However, it does not detect adversarial examples. Instead, this method merges semantically similar features (i.e., API call sequences) into a single representative feature. The evaluation experiments were performed on a dataset that includes API calls of benign and malware activity. The *Sequence Squeezing* method was also compared with other defense methods (e.g., adversarial training, adversarial signatures, RNN ensemble, Nearest Neighbor, defense sequence GAN) in four AML attacks (black-box gradient-based, white-box gradient-based, adaptive white-box attacker, and random perturbation). The experimental results concluded that sequence squeezing is the only method that balances low overhead and high robustness.

Demontis et al. [202] delved into the evasion of AML attacks by investigating the concept of bounding classifier sensitivity to feature changes to improve the classifier's security. The idea is to employ evenly-distributed feature weights to make the adversary alter more features to evade detection. The experiments were performed on an SVM classifier trained to detect Android malware. The authors concluded that the proposed approach enhances the classifier's security by about ten times when considering the number of modifications required to create a malware sample that evades detection.

Esmaeilpour et al. [203] proposed an environmentally sound classification approach that is robust against AML attacks without incorporating any defense method (reactive or proactive). The SVM classifier was deployed since the authors argued that due to the fact that learns for low-dimensional feature vectors is less sensitive to adversarial perturbations compared to DNN. Before extracting the robust features, the authors employed a pre-processing approach named spectrogram pre-processing that aimed to enhance the resilience of the SVM model against AML attacks. Moreover, the authors focused on the feature extraction process by evaluating several handcrafted features to conclude on the Speed Up Robust Features (SURF). Several experiments were carried out to assess the performance of the proposed classification approach in both original and adversarial examples (in black-box adversary knowledge). The authors concluded that CNN models could accomplish comparable performance in audio classification tasks, but they can be easily fooled by adversarial examples achieving more than 94% fooling rate, while the proposed approach reduced the fooling rate to 50%.

Han et al. [204] proposed a defense method based on feature reduction to protect an ML-based malicious network traffic detector against the gray and black-box AML attacks. The authors calculated the adversarial robustness scores of the features and reduced the feature set by deleting the features with the lowest adversarial robustness scores. A thorough evaluation was performed, where six ML/DL algorithms (two of them were unsupervised) were deployed, namely Logistic Regression (LR), MLP, DT, SVM, Deep Autoencoders (DAE), and Isolation Forest (IF). Furthermore, the proposed defense method was compared with adversarial training and feature selection defenses, and the results indicate that feature reduction is more effective in accomplishing better results in reducing the malicious Traffic Evasion Rate (MER), Probability Decline Rate (PDR), Malicious features Mimicry Rate (MMR). However, the authors did not evaluate whether the proposed defense affects the performance of the original samples.

Weerasinghe et al. [205] introduced a defense method for SVM classifiers to tackle poisoning and label flipping attacks based on Local Intrinsic Dimensionality (LID) (i.e., a measurement to characterize the outlierness of data samples). The authors proposed the K-LID estimation to distinguish the adversarial and original samples based on their traits, which was deployed to enhance the resistance of an SVM classifier against training data manipulations. Several experiments were performed on various real-world datasets (e.g., image, audio classification) to assess the performance of the K-LID SVM (i.e., testing against five different label-flipping attacks and one poisoning attack). The experiments concluded that the proposed defense could reduce the classification error rates by 10% on average.

Chen et al. [206] introduced a detection system for android malware, named KuafuDet that contains an adversarial detection mechanism (camouflage) to protect the system against poisoning attacks. The camouflage detection deploys similarity-based filtering analysis to detect malicious data distortions. KuafuDet were tested against three threat models (weak attacker, strong attacker, and sophisticated attacker) that differ in the way they generate the adversarial examples. The experimental results concluded that KuafuDet enhances the detection accuracy by at least 15%, reducing the false negatives.

2.4.2 Detection

In contrast to the Mitigation category, the Detection category focuses on reactive defenses whose purpose is to detect AML attacks (e.g., detection of adversarial examples) before they reach the ML/DL models.

2.4.2.1 Supervised/Semi-supervised Learning

The category Supervised/Semi-supervised Learning includes methods that detect AML deploying an auxiliary ML/DL model that has been trained using supervised or semi-supervised learning.

Pawlicki et al. [207] developed a detection method for AML attacks against ANN-NIDS. The authors specifically utilized four well-known adversarial attacks (C&W, FGSM, BIM, and PGD) to generate adversarial examples. The original samples (both malicious and benign) are labeled as "non-adversarial" based on the activation of each neuron of the ANN-IDS. The generated adversarial examples, along with the "non-adversarial", deployed to form a new adversarial dataset that contains the labels adversarial and non-adversarial. Moreover, the authors deployed five ML algorithms (ANN, RF, KNN, SVM, AB) to detect whether a sample is related to an AML attack and evaluated them on the CIC-IDS dataset. The best results were accomplished with RF and k-Nearest Neighbors (KNN) algorithms that attained 99% recall for AML attacks.

Taheri et al. [208] presented a label-flipping attack (Silhouette-based Label Flipping) along with two defense methods (label-based semi-supervised and clustering-based semi-supervised defense) that applied on a CNN classifier. The authors deployed their approach on the Android malware detection field using three different datasets (i.e., Drebin, Contagio, and Genome). The experimental results concluded that using clustering-based semi-supervised along with RF feature selection, the accuracy improved by 19% in comparison with state-of-the-art defenses.

The authors of [209] proposed a detection method, named PDGAN, to defend poisoning attacks against Federated Learning models. The attacker's goal is to compromise the global model by uploading malicious updates from the poisoned local model. This is achieved by performing a label-flipping attack on the training dataset of the local model and computing the poisoned local updates. The PDGAN detection method implements a GAN on the server to reconstruct the training data. Then, the training data are fed to each participant model to get the labels of each sample and calculate the accuracy of the participants' models. First, the participant' models were classified as benign and attackers (based on a threshold value), and later the PDGAN considers only the updates that have been performed by the benign participants. The authors evaluate the PDGAN on the image classification field using two datasets (MNIST and Fashion-MNIST). The experimental results demonstrate that the proposed method can immediately reduce the poisoning accuracy from almost 90% to 3.1% and accomplish an overall accuracy of almost 90%.

Recent defenses in the audio domain are based on randomization schemes to mitigate the effects of AML attacks on the audio samples. The authors in [210] use a randomization technique based on recursive filters. The technique first transforms the audio waveform into the frequency domain, resulting in a signal spectrogram. In parallel, recursive filters that are incorporating randomness are applied in the original audio to produce a new spectrogram that is different from the original. Randomness is considered in the form of probabilistic distributions. Both spectrograms are fed into an ASR system which compares them and derives their transcribed sentences using the word error rate. In the same manner, the ASR system can detect changes if the original audio is malformed due to an AML attack. This ASR system technique provides protection against both high-frequency and low-frequency audio signals and is resilient to adaptive attacks. The evaluation is performed in the C&W and PsyImp [211] AML attacks using DeepSpeech as well as Lingvo ASR systems and demonstrates a good detection performance for different probabilistic distributions (e.g., Gamma, Uniform, Normal).

2.4.2.2 Distance-based

In the Distance-based category, the detection of AML attacks is performed by measuring the distance between adversarial and original samples.

Paudice et al. [212] proposed an outlier detection method for poisoning attacks. Particularly, the authors deployed a distance-based anomaly detection for the detection of adversarial training samples using a part of original (i.e., trusted) data samples. The evaluation was performed on two benchmark datasets in spam filtering and computer vision domains as well as on KNN and SVM ML algorithms. The results concluded that the proposed method is capable of mitigating poisoning attacks. Furthermore, the authors evaluated their detection method for label-flipping attacks; however, the results were not encouraging because the generated adversarial examples were closer to the original, and the proposed detection method failed to detect them effectively.

Meng et al. [213] presented a framework, named MagNet, for defending ANNs image classifiers against AML attacks that are independent of the target classifier and the deployed adversarial examples generation process (i.e., it is attack-agnostic). Mag-Net consists of two components, the detector and the reformer. The detector rejects a sample that exceeds the boundary, while the reformer finds an alternative example closer to the boundary and sends that to the ANN classifier. More specifically, MagNet models only the original samples and estimates the distance of the test samples and the boundary of the manifold of the original samples. The authors used various AML attacks to generate the adversarial examples (e.g., C&W, FGSM, DeepFool, etc.) as well as the MNIST and CIFAR-10 datasets. The results showed that MagNet significantly increased the accuracy against all the attacks, while the accuracy only on original samples was reduced by 3.8%.

Esmaeilpour et al. [214] proposed a detection method that explores subspaces of adversarial examples in the unitary vector domain and developed a detector based on LR to defend models on environmental sound classification field. Specifically, the authors calculated chordal distance eigenvalues of legitimate and adversarial examples, which were deployed to train the LR classifier. The experiments were performed on three datasets (ESC-10, ESC-50, and UrbanSound8k) using eight different types of AML attacks (e.g., FGSM, BIM, JSMA, label flipping, etc.). The results concluded that the proposed detector achieved high performance in detecting all the attacks as well as it out-performed other detectors (e.g., kernel density, Bayesian uncertainty, and local intrinsic dimensionality) in six out of eight adversarial attacks.

The authors in [215] introduced a distance-based method for the detection of adversarial inputs in audio samples. The method is based on a recognition model, which performs classification to first characterize the original audio, and then it receives the modified audio as input. The modified audio is generated using low-pass filtering mechanisms, where the spectrum, waveform, and accuracy rate of the audio signal are analyzed. If the difference between the modified and the original audio is sufficiently large, then the original audio sample is regarded as an adversarial example. Similarly, if the difference is small, the original audio sample is regarded as a normal sample. The method is evaluated on the Mozilla Common Voice dataset16, where the C&W attack is performed to generate the adversarial audio samples and the results concluded that it outperforms other detection methods, such as white noise, denoise down-sampling, and temporal dependency. Another defense approach for adversarial audio detection is illustrated in [216], where distance-based techniques were followed to perform the detection. The authors introduced a novel ASR framework that performs similarity calculation through binary classification. The similarity is measured using the Jaro-Winkler distance method [217]. The method uses multiple ASRs to achieve a higher accuracy reaching even up to 99.88%.

Yang et al. [218] proposed a detection method against AML attacks for ASR models. The proposed method relied on the temporal dependency in speech signals. The audio sample is split into portions, and the defense framework evaluates the transcription of the portion against the original sample. The portion transcription is based on different transformation functions, such as quantization, down-sampling, local smoothing, and auto-encoder reformation of audio signals. The authors verified the portions that are not similar using the Word Error Rate (WER) and the Character Error Rate (CER). The results showed that the proposed temporal dependency detection method could effectively detect adversarial audio examples.

A similar approach that detects audio AML attacks in ASR models is the WaveGuard framework [219]. WaveGuard uses transformation functions and analyses the ASR transcriptions of the original and transformed audio. These functions are leveraging Receiver Operating Characteristic (ROC) curves for the detection of AML attacks. WaveGuard is able to detect AML attacks with a variety of audio transformation functions.

2.4.2.3 Feature Squeezing

Xu et al. [201] introduced Feature Squeezing, a method to detect adversarial examples in the computer vision domain. The method is based on two-dimensionality reduction techniques: i) Color bit depth reduction (from 24-bit to 8-bit) and ii) spatial smoothing (i.e., blur). The first technique is based on the fact that the image's colors are unrelated to the classification; hence they will not affect the model's classification accuracy when reduced. Spatial smoothing, on the other hand, employs local smoothing and non-local smoothing to reduce the image's noise. During the Feature Squeezing process, two versions of the input image are produced based on the two aforementioned techniques. A DNN classifies the original image and the two generated images, and their outputs are compared using the L_1 metric. If L_1 exceeds a threshold, the image is classified as adversarial and discarded.

2.4.2.4 Statistical

Grosse et al. [220] proposed a detection method based on Statistical tests for adversarial examples. Two statistical distance metrics (Maximum Mean Discrepancy (MMD) and Energy Distance (ED)) were employed to measure the distance of adversarial and original samples, namely to identify their differences. The authors evaluated their method on four classifiers (ANN, DT, LR, SVM), three different domains, such as image classification (MNIST), malware detection (Drebin), and cancer detection (MicroRNA), as well as four AML attacks (FGSM, JSMA, SVM attack, and Decision Tree attack) and concluded that it could effectively detect the adversarial examples.

Table 4 summarizes all the above-mentioned defense methods based on key traits, such as year, deployed attack type (i.e., poisoning or evasion), attacker's knowledge (i.e., black/white/gray-box), application domain, evaluated ML/DL algorithms, as well as the employed defense method (e.g., adversarial training, ensemble methods, pre-processing, etc.).

Table 4 Summary of AML Defense Methods

Paper	**Year**	**Attack Type**	**Attacker's Knowledge**	**Domain**	**Algorithm**	**Defense Method**
Anthi et al.	2021	Poisoning	Gray-box	Cybersecurity -ICS	RF/J48	Adversarial Training
Vitorino et al.	2022	Poisoning	Gray-box	Cybersecurity -IoT	MLP/RF	Adversarial Training
Khamis et al.	2020	Poisoning	White-box	Cybersecurity -Network	DNN/RF/AB/ NB/SVM	Adversarial Training
Khamis et al.	2020	Poisoning	White-box	Cybersecurity -Network	CNN/RNN	Adversarial Training
Apruzzese et al.	2020	Poisoning	Gray-box	Cybersecurity -Botnet	RF/WnD	Adversarial Training
Fu et al.	2021	Poisoning	White-box	Cybersecurity -Network	CNN/LSTM/ GRU	Adversarial Training
Rashid et al.	2022	Poisoning	N/A	Cybersecurity -IoT	MLP/DT/RF	Adversarial Training
Goodfellow et al.	2014	Poisoning	N/A	Image Classification	DNN	Adversarial Training
Huang et al.	2015	Poisoning	N/A	Image Classification	DNN	Adversarial Training
Madry et al.	2017	Poisoning	Black/White -box	Image Classification	DNN	Adversarial Training
Sankaranarayanan et al.	2018	Poisoning	N/A	Image Classification	DNN	Adversarial Training
Shafahi et al.	2019	Poisoning	N/A	Image Classification	DNN	Adversarial Training
Xie et al.	2019	Poisoning	Black/White -box	Image Classification	CNN	Adversarial Training
Tramèr et al.	2017	Poisoning	Black/White -box	Image Classification	CNN/FCN	Adversarial Training
Lal et al.	2021	Poisoning	Black/White -box	Image Classification	CNN	Adversarial Training & Feature Reduction

Ding et al.	2021	Poisoning	N/A	Visual Context Prediction	cGAN	Adversarial Training
Xu et al.	2021	Poisoning	N/A	Image Classification	DNN	Adversarial Training
Liu et al.	2020	Poisoning	N/A	NLP	BERT/RoBERTa	Adversarial Training
Yoo et al.	2021	Poisoning	N/A	Text Classification	BERT/RoBERTa	Adversarial Training
Pan et al.	2021	Poisoning	White-box	Text Classification	BERT/RoBERTa	Adversarial Training
Kitada et al.	2021	Poisoning	N/A	NLP	RNN	Adversarial Training
Morandi et al.	2022	Poisoning	Black/White-box	NLP	BERT	Adversarial Training
Pal et al.	2021	Poisoning	Black/White-box	Speech Recognition	DNN	Adversarial Training
Sonal et al.	2022	Poisoning	White-box	Speech Recognition	CNN	Adversarial Training
Papernot et al.	2016	Poisoning	N/A	Image Classification	DNN	Defensive Distillation
Goldblum	2020	Poisoning	N/A	Image Classification	DNN	Defensive Distillation
Soll et al.	2019	Poisoning	N/A	Text Classification	CNN	Defensive Distillation
Apruzzese et al.	2020	Evasion	N/A	Cybersecurity-Botnet	RF	Defensive Distillation
Apruzzese et al.	2020	Evasion	Gray-box	Cybersecurity-Botnet	DT/RF/MLP/AB/WnD	Ensemble Methods
Jiang et al.	2022	Evasion	Black-box	Cybersecurity-IoT	LSTM	Ensemble Methods
Sonal et al.	2021	Evasion	White-box	Speech Recognition	DNN	Ensemble Methods
Rosenberg et al.	2019	Evasion	Black/White-box	Cybersecurity-Network	RNN	Pre-processing
Demontis et al.	2017	Evasion	N/A	Cybersecurity-Android	SVM	Pre-processing
Esmaeilpour et al.	2019	Evasion	Black-box	Audio Classification	SVM	Pre-processing
Han et al.	2021	Evasion	Gray/Black-box	Cybersecurity-Network	LR/MLP/DT/SVM/DAE/IF	Pre-processing
Weerasinghe et al.	2021	Poisoning	Gray-box	Image & Audio Classification	SVM	Pre-processing
Chen et al.	2018	Poisoning	Gray-box	Cybersecurity	SVM	Pre-processing
Pawlick et al.	2020	Poisoning	White-box	Cybersecurity-Network	ANN/RF/KNN/SVM/AB	Supervised Learning
Tao et al.	2021	Poisoning	N/A	Speech Recognition	DNN	Supervised Learning
Taheri et al.	2020	Poisoning	Black-box	Cybersecurity-Android	CNN	Semi-supervised Learning

Zhao et al.	2019	Poisoning	White-box	Image Classification	Federated Learning	GAN
Paudice et al.	2018	Poisoning	N/A	Spam Filtering & Computer Vision	KNN/SVM	Distance-based Detection
Meng et al.	2017	Poisoning	Black-box	Image Classification	ANN	Distance-based Detection
Esmaeilpour et al.	2020	Poisoning & Evasion	N/A	Audio Classification	CNN/SVM	Distance-based Detection
Kwon et al.	2020	Evasion	Gray-box	Audio Classification	DNN	Distance-based Detection
Zeng et al.	2019	Evasion	White/Black -box	Speech Recognition	DNN	Distance-based Detection
Yang et al.	2018	Evasion	N/A	Speech Recognition	DNN	Distance-based Detection
Husssain et al.	2021	Evasion	White-box	Speech Recognition	DNN	Distance-based Detection
Xu et al.	2017	Poisoning	White-box	Audio Classification	DNN	Feature Squeezing
Grosse et al.	2017	Poisoning & Evasion	Black/White -box	Image & Malware & Cancer Classification	ANN/DT/LR/ SVM	Statistical

2.5 Evaluation of Defense Methods

This section aims to provide insights regarding the efficient evaluation of defenses (both mitigation and detection methods) against AML attacks. To this end, we gather from the literature and present datasets that contain both original and adversarial samples (hereafter AML datasets) as well as evaluation metrics that have been specifically created to assess the robustness of AI systems or defense methods against AML attacks. It should be noted that common evaluation metrics such as accuracy, precision, recall, and F1-score are not considered in this section since they are not appropriate for measuring the robustness of AI systems in adversarial settings.

2.5.1 Dataset

In this section, the AML datasets that have been introduced by existing works and are publicly available to evaluate ML/DL models' robustness were gathered. To the best of our knowledge, only seven AML datasets are publicly available, six of them belong to the computer vision domain and one to the sound classification. These datasets are presented below:

1. DAmageNet [221] is an AML dataset that provides black-box AML attack images. It contains 96,020 adversarial samples generated from the original samples of the ImageNet dataset. DAmageNet is appropriate for testing and improving the robustness of DNN in image classification tasks[20].
2. In [222] four variations of the CIFAR dataset have been proposed in order for the authors to demonstrate that adversarial samples are a result of non-robust features (i.e., features that can be easily predicted but are incomprehensible to humans). The datasets are publicly available[21] and contain both robust and non-robust features (i.e., both original and adversarial samples) appropriate for image classification tasks.

[20] http://www.pami.sjtu.edu.cn/Show/56/122
[21] https://github.com/MadryLab/constructed-datasets

3. The authors of [170] generated two AML datasets based on the CIFAR-10 and CIFAR-100 using PGD AML attack to produce the adversarial examples[22]. These datasets can be deployed to develop robust image classification models or evaluate the models' robustness against PGD attacks.
4. The APRICOT dataset [223] was developed to deceive object detectors. It is comprised of 1000 images of printed adversarial patches photographed in real-world scenes (i.e., public locations)[23]. The samples are generated to represent various viewing angles, positions, light, and distance conditions. Furthermore, this dataset can be used to assess the robustness of object detectors as well as to facilitate researchers in developing methods for flagging adversarial patches.
2. The authors of [224] introduced two adversarial datasets based on ImageNet, named ImageNet-A and ImageNet-O[24]. ImageNet-A is comprised of real-world adversarially filtered images that can be employed to assess image classifiers' robustness, while ImageNet-O contains adversarially filtered images to evaluate out-of-distribution detection models.
3. In [225], the authors proposed a simulation tool for generating synthetic datasets with realistic adversarial samples using the CARLA simulator[25]. This dataset can be deployed to test the adversarial robustness of frameworks in object detection tasks and similar research fields such as autonomous driving.
4. Adv-ESC (Adversarial Environmental Sound Classification) dataset [226] is the first environmental sound dataset equipped with adversarial examples, which can serve as a benchmark evaluation dataset to train robust sound classification models. Adv-ESC is based on two environmental sound classification datasets (ESC-10 and DCASE task 1A) and is comprised of adversarial examples generated by FGSM, PGD, and BIM methods.

It has been observed that only a few AML datasets are publicly available, where the vast majority can only be applied to the computer vision domain and, more specifically, to image classification and object detection tasks. There is a clear need for more AML datasets that can also be applied to other domains, such as cybersecurity, where AML attacks are on the rise. Hence, future research on AML should also consider the introduction of benchmark AML datasets in various domains to facilitate researchers evaluating their defense methods and comparing them with existing works on common ground.

2.5.2 Evaluation Metrics

This section elaborates on metrics that existing works utilized to measure the efficacy of their defense solutions against AML attacks. Most works employ common metrics to evaluate the robustness against AML attacks (e.g., accuracy, F1-score, AUC); however, a few introduce new metrics. Here, we discuss these metrics.

Malicious traffic Evasion Rate (MER) [204]**.** It measures how much original malicious network traffic can be evasive. MER is the percentage of undetected mutated malicious traffic to originally detectable traffic.

$$MER = 1 - \frac{Pumt\_mal}{Pdt}$$

Detection Evasion Rate (DER) [204]**.** This metric is similar to MER but measures how much of the mutated traffic can be evasive. DER also measures whether the adversarially generated traffic is

[22] https://github.com/ashafahi/free adv train/tree/master/datasets
[23] https://apricot.mitre.org/download/
[24] https://github.com/hendrycks/natural-adv-examples
[25] https://carla.org

detected as malicious. DER is the percentage of undetected mutated traffic to originally detectable traffic.

$$DER = 1 - \frac{Pumt}{Pdt}$$

Malicious Probability Decline Rate (PDR) [204]**.** PDR measures the decline rate of the malicious probabilities produced by the classifier.

$$PDR = 1 - E_{(f,fn) \in (Fmal,Fnmal)} \frac{C(fn)}{C(f)}$$

Malicious features Mimicry Rate (MMR) [204]**.** This metric measures the extent to which adversarial crafted features are close to malicious features extracted from mutated traffic.

$$MMR = 1 - E_{(f,fn,fa) \in (Fmal,Fnmal,Fadv)} \frac{L(fn, fa)}{L(f, fa)}$$

Fooling Rate (FR) [203]**.** This metric is based on the error rate and computes the ratio of successful fooling adversarial samples (SAS) over the total number of adversarial samples (TAS).

$$FR = \frac{SAS}{TAS}$$

Attack Success Rate (ASR) [227]**.** The ratio of misclassified adversarial samples (MAS) to the total number of adversarial samples (TAS).

$$ASR = \frac{MAS}{TAS}$$

Attack Severity (AS) [196]**.** AS measures the efficacy of an evasion AML attack considering the detection rate (DR) before and after the attack.

$$AS = 1 - \frac{DR\_after}{DR\_before}$$

2.6 Discussion

Even though AML has received significant attention in the last five years and new AML attacks are constantly introduced or existing attacks are applied to different fields, the defense methods against AML attacks are still in their infancy. Based on the surveyed articles limited research is available regarding mitigating or detecting AML attacks.

Most of the existing defense methods that surveyed in this thesis focus on improving the resiliency of ML/DL models (i.e., on the mitigation category) against AML attacks rather than detecting them. More specifically, 74% of the surveyed defenses deal with the mitigation of AML attacks. This can be attributed to the fact that adversarial training is one of the first defense methods that are still extremely popular because researchers strive to introduce adversarial training methods which do not affect the model's performance in non-adversarial settings. Furthermore, only 8% of the surveyed defenses are attack-agnostic, and 4% are domain-agnostic, indicating that most of the existing defenses cannot cope with multiple AML attacks neither can be applied to more than one domain. Most previous works were not validated using appropriate evaluation metrics since the works that have been evaluated using AML evaluation metrics are only 14%. In addition, limited information is publicly shared by the authors of existing works to facilitate the scientific community to reproduce their results and compare them with future works considering that only a few works share the code (38%) and only one its deployed dataset (2%). Last but not least, the defenses against AML attacks that target NLP and audio applications have not been adequately studied since only a few defenses exist in these domains.

The taxonomy introduced in this work revealed several gaps and open problems in the existing literature that paved the way for identifying some directions for future research in the AML defense domain. ML/DL models are utilized almost in every technological domain. Consequently, the surface of AML attacks is massive; hence, it is crucial to develop domain-agnostic defenses (i.e., a defense that can be applied to several domains). From the surveyed defenses, only two works ([212] and [220]) evaluated their defenses on more than one domain. Thus, one direction for future research could focus on developing domain-agnostic defenses. Towards the same direction, AML attacks are evolving as new attacks are continuously emerging. Therefore, it is crucial to develop defense methods that are attack-agnostic. However, currently, only a few existing defense methods are attack-agnostic (e.g., [200], [213], [214], [220]). Hence, a second direction for future research is designing new attack-agnostic defenses. Moreover, since existing defenses mostly deal with mitigating AML attacks, there is a gap in defense methods that concentrate on detecting AML attacks. To this end, a possible third future research direction could focus on proposing innovative detection methods for AML attacks or hybrid solutions that will combine both mitigation and detection. Finally, evaluating the robustness of ML/DL models is a critical issue, and proper evaluation metrics and datasets for AML attacks should be considered. Thus, a fourth future research direction could emphasize proposing new AML evaluation metrics and datasets.

3. An Ensemble Learning Methodology for Detecting Phishing Email Attacks

3.1 Introduction

Phishing email detection have drawn the focus of academia for more than a decade. Over the years, scientists have proposed various ML-based detection methods to tackle phishing email attacks. Although substantial research exists in the literature, numerous limitations and gaps have been identified, which are presented in section 3.2.3. These pitfalls include the utilization of a narrow set of classifiers (i.e., most of the existing works assess the same ML algorithms). Additionally, even though phishing detection is an imbalanced classification problem, balanced datasets are deployed evaluated using inadequate evaluation metrics. Moreover, limited available data sources are publicly available which leads to dataset issues. The majority of existing works employ obsolete phishing emails that do not represent current attack trends. Furthermore, the existing works mostly focus either on the email's contents (i.e., attributes of an email that represent its structure, like headers, URLs, syntax, attachments, etc.) or on the email's body text (applying Natural Language Processing and text analysis) to extract features that can lead in the detection of phishing emails. All the above mentioned drawbacks affect the applicability of the existing works in real-life conditions letting individuals and organizations unprotected.

To the best of our knowledge, there are no recent work in the literature that deploys both the emails' contents and body text to generate a compact, yet thorough representation of emails in a format that can be processed by ML algorithms. In addition, Ensemble Learning (e.g., the combination of two or more ML algorithms) is a popular ML method that has enhanced the results in numerous research domains, to name a few, phishing web page detection [13] and fraud detection [14]; nevertheless, it has not been examined in depth in phishing email detection. These observations along with the evolution of phishing email attacks, which has rendered many of the existing detection solutions obsolete, were the motivation of proposing an innovative methodology for the detection of phishing emails.

In this regard, detecting a phishing email focusing only on the email's contents or body text can be judged as insufficient, because latest phishing emails are too complicated and more like benign, thus their detection is far more challenging comparing to older phishing emails. This thesis is based on the assumption that hybrid features (i.e., fusion of content and body text features) contain rich information that sufficiently depicts the phishing emails representing even the smallest deviations. Despite the fact that the main concept is simple, a challenging task is to efficiently manage the hybrid features without letting information (of the features) unexploited. Furthermore, it is also assumed that traditional ML methods cannot effectively cope with hybrid features to enhance phishing email detection performance since, the structure of content-based and text-based features is different, yet the traditional ML algorithms handle them equally. On the other hand, Ensemble Learning is able to process the hybrid features separately via the implementation of two algorithms, one algorithm that will handle the content-based features and one that will handle the text-based features. In this way, it is believed that Ensemble Learning will result in improved detection results. Based on the above mentioned, this work aspires to answer the following research question:

Can the combination of Ensemble Learning with hybrid features, overall, enhance the phishing email detection performance? (RQ2)

To answer this question, this Chapter makes the following contribution:

An innovative Ensemble Learning methodology for phishing email detection that employs hybrid email features and improves the detection performance in phishing email detection domain.

Particularly, the introduced data-driven Ensemble Learning methodology, named HELPHED (Hybrid Ensemble Learning PHishing Email Detection), implements two different Ensemble Learning methods (stacking and soft-voting) and hybrid features obtained both from the contents and body text of emails (i.e., the email's message that is shown to the recipient). Using hybrid features the email is converted into a compact, yet thorough representation that can be efficiently interpreted by the Ensemble Learning methods to achieve high detection performance. HELPHED methodology includes in total 6 stages. The first stage is the *Email Parsing*, where the email's contents (i.e., body text and headers) are divided and stored in arrays in order to facilitate their processing in the next steps. The second stage performs the *Content-based Feature Extraction* process that extracts 22 features from the emails' contents. These features are divided into four categories: body, syntactic, header, and URL. Next, the *Pre-processing* stage is performed in parallel with content-based feature extraction. This stage prepares the email's body text for the next stage, namely *Textual Feature Extraction* that obtains the text-based features using the Word2Vec method. In stage 5, the *Feature Selection* takes place, where the Mutual Information method is employed to select the most informative features of the hybrid feature sets. Finally, the methodology concludes with the emails' binary classification between benign and phishing, namely the *Ensemble Classification* stage. In this stage two methods are proposed, Method 1 -Stacking Ensemble Learning and Method 2 - Soft Voting Ensemble Learning. A novelty of the proposed methodology is that these methods are comprised of two ML algorithms to process the hybrid features separately, yet in parallel to exploit all the information that both feature categories can offer.

The rest of the chapter is organized in the next sections: Section 3.2 presents the related work in the detection of phishing emails using ML/DL methods. Section 3.3 introduces the Ensemble Learning architecture, describing in detail the different stages of the methodology. Section 3.4 discusses the approach that was followed for the evaluation of the methodology, also proposing a set of rules for the proper evaluation to avoid misleading results, presents the deployed dataset and metrics, elaborates on the experimental results, and compares the methodology with state-of-the-art existing works. At this point, it should be noted that parts of this chapter have been published in [31] and [33].

3.2 Related Work

Given that emails cannot be directly processed by ML algorithms, feature engineering is imperative to convert an email into a compact representation that can be interpreted by an ML or DL algorithm. Existing research mainly focus on content or text-based features to distinguish benign from phishing emails. Based on this observation, the existing works on ML/DL phishing email detection are grouped on two categories: detection of phishing emails using content-based features and detection of phishing emails using text-based features.

3.2.1 Detection of phishing emails using content-based features

Ma et al. [57] were among the first that introduced the concept of hybrid feature extraction for phishing email detection. The hybrid features obtained from three categories: content features (e.g., domain-specific keywords), orthographic features (e.g., style characteristics), and derived features (e.g., a combination of the existing content and orthographic features). The authors extracted seven features in total from the aforementioned categories and fed them in a decision tree (C4.5) classifier, which accomplished high detection accuracy. A major limitation of this work is that it did not utilize the appropriate evaluation metrics (i.e., only the classification accuracy has been deployed) to capture the detection performance on the deployed imbalanced dataset (only 7% of emails were phishing). Namely, the results of [57] are biased toward the majority class [58]. In [59] the authors also presented a hybrid approach that combines content-based and behavior-based features. Seven features were obtained in total, and a Bayesian Network classifier was deployed to predict whether an email is phishing or benign. The authors tested their approach on the Nazario's dataset [60] with emails from 2004-2007 and their approach produced promising results with 92-96% detection accuracy. The authors heavily relied on domain and URL features (as four out of the seven features are related with the domain and URL), which renders this work insufficient on recent phishing emails attacks that are more sophisticated and can easily evade these features. Both [57] and [59] have been proposed more than a decade ago, hence their efficacy on newer phishing emails is disputable. This work also embraces the concept of hybrid features, but instead of the integration of behavior-based and content-based features, content-based features were combined with text-based features.

Moradpoor et al. [61] deployed an Artificial Neural Network (ANN) classifier to classify phishing emails. The introduced model includes six components: 1) The Emails component, which contains the dataset (namely, the phishing and benign emails), 2) the Email Classifier component, which classifies the emails into phishing and benign, 3) the Email Parser component, which extracts four features from the emails that consider the email's contents, parts, encoding, and text, 4) the Email Sanitizer component, which performs a text cleansing process (e.g., lowercase conversion, removing special characters and stopwords, etc.), 5) the Email Vectorizer component, which deploys Word2Vec to assign each unique word to a corresponding vector and calculates the average value of the vectors, and 6) the Neural Network Model component, which utilizes an ANN classifier to predict whether the email is phishing or benign. The evaluation performed on public email datasets (e.g., [60], [62]) and the classifier accomplished 92.2% detection accuracy. This work has several limitations, as the implementation of the aforementioned components (i.e. Email Classifier) were not thoroughly discussed and the features extracted from the emails are not sufficient (i.e., the authors only extract four features from the email's contents), which results in lower detection accuracy than other works in the literature. This issue was addressed and increased the accuracy by extracting more representative content-based features and exploiting all the word vectors by using them as input in the classifier instead of calculating their average value. In [63], the authors extracted in total 15 features (identified from the literature) from the email's contents. Some of these features are: URLs including an IP address, hyperlinks, and disparities between "href" attribute, occurrence of JavaScript code, number of dots in domain name, HTML email, etc. A Random Forest classifier was deployed to perform the classification based on 10-fold cross validation and achieved 99.7% classification accuracy and 98.45% F1-score. This work accomplished comparing results, however, the authors deployed an unknown (i.e., it cannot be validated) and imbalanced dataset that contains a small number of emails (1,800 benign and 200 phishing), as well as they did not deploy appropriate evaluation metrics for imbalanced data.

Smadi et al. [64] proved that the pre-processing phase can enhance the efficacy of the classifier. Particularly, the authors divided the feature extraction process in three phases and in each phase extracted different features from different email parts (e.g., In phase 1, two features have been extracted from the headers, in phase 2, four features have been extracted from the email content, etc.). The authors extracted in total 23 behavior and content-based features from a dataset that have been created using 4,559 phishing emails obtained from [60] and 4,559 benign emails from [62]. The paper concludes that the Random Forest classifier accomplished the best accuracy (98.87%). Another work that focuses on the URLs that included in the emails proposed in [65]. In this work, the authors presented PhishStorm, where it was defined the term intra-URL relatedness which practically investigates the relationship between the words that comprising the various fields of a URL (namely, the domain name and the remaining of the URL). 12 features were extracted from the URLs that were used to evaluate seven supervised ML classifiers. The evaluation results concluded that the Random Forest classifier accomplished the higher detection accuracy (94.91%). Both [64] and [65] have two main limitations. First, these works focus on the URLs of the emails; hence, in case that there are no URLs (e.g., a phishing email that delivers a malicious file), these approaches will not be effective. Second, in [64], the authors used publicly available datasets, but they did not mention the emails' creation date, while in [65] the authors tested their approach only on URLs, hence it could not be considered as a phishing email solution.

Furthermore, additional content-based works (e.g., [66], [67], [68]) have been proposed the previous decade (before 2010) to address the phishing email challenges. These works mostly examine URLs, vocabulary, natural structural characteristics (e.g., From, To, Subject fields, etc.), presence of JavaScript in the emails, body and subject fields focusing on the domains, the words, and the characters they contain. These works even though they attained good results once, it is not proven that they are able to follow the advancement of phishing emails (i.e., a re-evaluation is needed with newer phishing and benign emails).

3.2.2 Detection of phishing emails using text-based features

An interesting observation is that the older works (before 2015) on phishing email detection primarily focus on content-based features, while the most recent works focus on textual features. This observation can be attributed to the recent evolution of NLP methods. Artificial Intelligence (AI) and NLP have contributed to the robust detection of phishing emails by using only the textual information that appears in them. Past works have employed syntactic, semantic, and contextual features to detect phishing emails. A recently published work proposed in [69], where the authors deployed NLP techniques along with ANN for the emails' classification. Particularly, the TF-IDF method was employed to create the text-based feature set and a Graph Convolution Network was used as a classifier. The experiments executed on a dataset [70] that includes 3,685 phishing and 4,894 benign emails. The authors used 3-fold cross-validation and the classifier reached 98.2% accuracy. A drawback of [69] is that the authors did not consider a realistic evaluation scenario in which the number of phishing emails is much less than the benign, as well as the deployed dataset, is comprised of old phishing emails that fail to capture the current trends of phishing email attacks.

The authors of [71] extracted distinctive features from the phishing emails' body text. Gualberto et al. considered the challenges of the sparsity, "the curse of dimensionality", as well as a portion of the text context that was included in the extracted feature set. Various numbers of features were employed to assess the efficacy of the proposed approach and the best outcomes were accomplished with 10 features, where the XGBoost algorithm reached a 99.95% success rate. An extension of [71] presented in [72], where it was proposed a multi-stage solution for the detection of phishing emails that employs two methods. In method 1, the authors utilized Chi-Square statistics and Mutual Information to perform dimensionality reduction, while in method 2, it was deployed Principal Component Analysis (PCA) and Latent Semantic Analysis (LSA). The best overall performance, namely 100% accuracy and 100% F1 score, was achieved by method 2 with LSA and the XGBoost algorithm. The experiments were performed on the Spam Assassin [73] and Nazario [60] datasets, employing 4,150 benign and 2,279 phishing emails. Both works deploy the same dataset to assess their approaches and the results were comparable; however, in [71] cross-validation was used for the evaluation, while [72] was evaluated in a dataset separated by 70% for training and 30% for testing. A major drawback of these works is that they did not take into account the phishing emails' growth, since the deployed phishing emails are old.

In [74], the authors presented a detailed study on phishing attacks and proposed THEMIS; a framework for the detection of phishing emails, which employs Recurrent Convolutional Neural Networks (RCNN). They divided the email into multiple levels, namely char-level and word-level of the header and char-level and word-level of the body, and the Word2Vec method [75] was used to acquire the vector sequences from the aforementioned levels. THEMIS accomplished 99.848% detection accuracy with 0.043% false positive rate on the IWSPA 2018 dataset. Towards the same direction, Hiransha et al. [76] proposed a deep learning approach for the detection of phishing emails relying only on the emails' textual information. The authors combined Convolutional Neural Networks along with Keras Word Embedding to build their detection model. The evaluation was performed on a dataset that contains the email headers and on a dataset without email headers that originated from the IWSPA 2018 dataset. The results indicate that the model performs better when the email headers are not considered, accomplishing 96.8% accuracy. Both [74] and [76] achieved promising results by focusing only on the emails' textual information, however, a common limitation is that the authors did not test their models using newer emails to evaluate whether they can capture the evolution of phishing email attack. Moreover, in [76] the authors deploy an imbalanced dataset and only the classification accuracy to measure the classifier's performance, which is inappropriate.

SAFE-PC [77] is a methodology for detecting new phishing campaigns using NLP methods to acquire text-based features. In particular, five features were extracted from a dataset of a university: i) structural features in the email, ii) proper noun organization names and their types, iii) commonly appeared phishing words from the utilized dataset and their synonyms, iv) words associated with the university, and v) most used phishing words and their synonyms. Then, a feature engineering process is performed to thwart common phishing strategies, and NLP techniques, such as Named Entity Recognition (NER), Free-base, and synonym substitution are employed to construct the final feature set. Finally, the classification is performed via a RUSBoost algorithm, which integrates boosting and data sampling, with various ML algorithms (i.e., Naive Bayes, decision trees, and MLP). SAFE-PC detected 71% of the phishing emails that had been erroneously classified by Sophos and achieved 15% FPR. Also, it surpassed the SpamAssassin antispam platform which had a detection rate of less than 10%.

The authors in [78] extracted in total 26 features that come from word counts, punctuation, and stopwords. The word-based features were used to compare the performance of various ML classifiers. The best performance was reached by SVM, which achieved 83% TPR and 96% FPR. The main drawback of both [77] and [78] is the weak feature sets that mainly relied on stopwords, punctuation counts, variations of words, and different ratios among the features, which can be effortlessly bypassed by attackers.

An approach that employs Recurrent Neural Networks (RNNs) to detect phishing emails utilizing only text-based features from the email introduced in [79]. The classification was performed via a word tokenization method, while the assessment was based on two datasets (which have the same benign emails and different phishing emails) that were used to compare the RNN classifier with a "textAnalysis" model [80] and a Dynamic Markov Chain (DMC) model [81]. [79]outperformed [80]; however, it reached worst results than [81]. Particularly it achieved 96.74-98.91% accuracy and 96.71-98.63% F1-score on two balanced datasets. The limitation of this approach appears in the evaluation since the authors did not provide sufficient information about the deployed phishing emails (e.g., they did not discuss the origin date), therefore it is not sure if this work can detect new phishing emails, as well as the authors did not base their evaluation on a real use case scenario using imbalance data.

Unnithan et al. [82] combined domain-level and text-based features. In particular, the authors deployed the TF-IDF method to extract the text features, which was used as input to the ML classifiers along with the domain-level features (i.e., the most commonly appeared words along with a list of special characters). For the assessment, the IWSPA dataset was employed, where the Logistic Regression classifier attained the best F1-score (98%). The evaluation data of [82] are obsolete since the deployed dataset mostly contains outdated phishing emails. A drawback of [82] is that the domain-level features are susceptible, meaning that they can easily be evaded by attackers and hence, reduce the efficacy of the approach. In another work, Unnithan et al. [83] compared the TF-IDF and Doc2Vec (a word embedding technique that is based on Word2Vec) methods for the extraction of text-based features from emails. The TF-IDF and Doc2Vec features are used as input to various ML algorithms, such as Naive Bayes, Logistic Regression, Decision Tree, AdaBoost, SVM, K-nearest Neighbor, and Random Forest, to detect the phishing emails. The IWSPA dataset was deployed to evaluate the classifiers' efficacy based on the detection accuracy metric. The paper concludes that the Doc2Vec contributes to better detection performance. A major limitation of [83] is that the authors relied their evaluation only on the accuracy metric that is inappropriate for imbalanced data, as well as the highest accuracy that was obtained by Doc2Vec and SVM (i.e., 88.4%) is not very good compared with the existing works.

3.2.3 Limitations of existing works

Although important progress has been made in phishing email detection concerning the detection accuracy there are still numerous drawbacks in the literature that have not been tackled by existing detection methods [15]. Regarding the existing works that employ content-based features, the most significant drawbacks are: i) outdated datasets (mainly since newer methods did not take into account content-based features), ii) insufficient features, and iii) poor evaluation experiments (e.g., limited evaluation metrics). Whilst, the major limitations of the existing works that focus on the textual features are i) limited evaluation metrics and samples, ii) limited experiments on newer phishing emails (i.e., the advancement of phishing emails is not taken into account), iii) the generalization of the results on newer phishing emails has not been examined, and iv) non-representative textual features. Another important limitation that identified in the literature is that existing detection solutions did not deploy different dataset sources to evaluate their detection performance.

Moreover, Ensemble Learning has achieved significant results in other research fields (e.g., [84], [85]); however, it has not been deployed yet to tackle phishing emails. More specifically, a limited number of works in the literature ([83], [86], [87]) utilize Ensemble Learning methods, which have several disadvantages. For example, the evaluation datasets contain a small number of old phishing emails, the utilized features are not enough to represent the emails, and the evaluation metrics are insufficient (i.e., the AUC and accuracy were mostly used. Thus, there is a major gap regarding the applicability of Ensemble Learning in the detection of recent phishing email attacks.

3.3 Methodology Overview

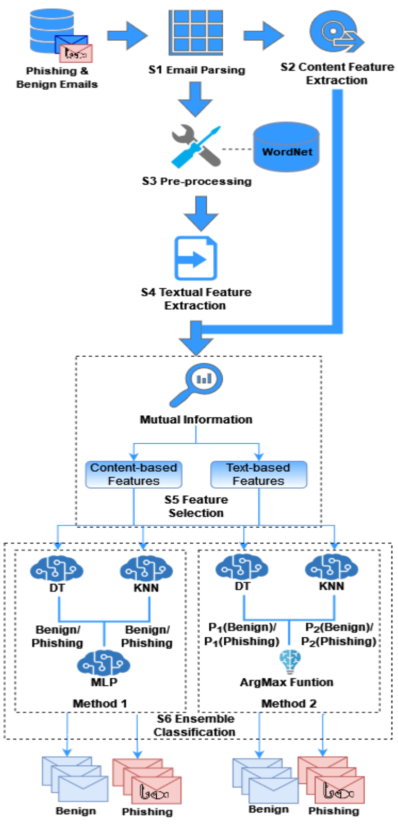


Figure 3-1 Phishing email detection methodology

Figure 3-1 depicts the methodology's stages, whose main objective relies mostly on identifying representative features of phishing emails and parsing them to novel Ensemble ML algorithms to evaluate later their detection performance on realistic email samples based on a real-life scenario. More specifically, the methodology that designed in this thesis begins with the Email Parsing stage (S1), where the header and the body fields of an email are split and parsed to a vector structure as well as they are associated with the email's respective class according to the collection from which it is originated (e.g., phishing or benign). Following, the Content-based Feature Extraction stage (S2) extracts the content-based features from the emails without further pre-processing. The content-based features include several parts of emails, such as Body, Syntactic, Header, and URL features. The Pre-processing stage (S3) takes place after the content-based features have been obtained in which the body text of emails is converted into a uniform format. This stage includes a cleansing process step, where the texts are transformed into lowercase, the special characters, stop words, and punctuation marks are deleted, and the hyperlinks are converted to a consistent string. Then, the lemmatization process is employed in the clean texts to consolidate the various inflected word forms and treat them as individual words. Following, the Textual Feature Extraction stage (S4) utilizes the Word2Vec method [75] to automatically extract the textual features. The novelty of this approach is that both content-based and text-based features were extracted from the emails and merged into a robust hybrid feature set to identify even the slightest variations between phishing and benign emails and hence offering an improved solution for the detection of the newer sophisticated phishing emails. The Feature Selection (S5) is the final stage prior to the classification, where the redundant features of both categories are identified by the Mutual Information method and excluded from the hybrid feature set. In this way, only the most informative features will be used in the classification stage. The Ensemble Classification stage (S6) of the methodology deploys the selected hybrid features, which provide a comprehensive representation of emails to feed the ML algorithms for the detection of phishing emails by predicting the email's class (namely, phishing or benign). To provide a more comprehensive approach two methods have been considered. Particularly, Method 1 implements a Stacking Ensemble Classifier, which is comprised of 2-tiers; Tier-1 employs the DT and KNN ML algorithms to perform the initial classification and Tier-2 employs the MLP ANN algorithm, which is based on the classification results of Tier-1 to detect whether an email is phishing or benign. While Method 2 implements a Soft-voting Ensemble Classifier that employs in Tier-1 the DT and KNN algorithms for the initial classification of the emails and in Tier-2 a voting function to detect the phishing emails. Another innovation of this methodology is that using Ensemble Learning the processing of the hybrid features separately was achieved, namely using different ML algorithms to process the content-based and the text-based features. In this way, the ML algorithms that provide the best classification results when parsing each feature set are deployed.

3.3.1 Email Parsing

In the Email Parsing stage (S1 of Figure 3-1), the emails' header and message body fields are split and stored in separated arrays. This stage aims at the organization of emails in arrays to be processed faster in the next stages of the methodology (i.e., S2-S6). The parsing operation springs from the fact that emails' format is intricate. Particularly, the email's body text contains tons of information that can facilitate researchers and security experts propose methods and tools for the detection of phishing emails. The main trait of body text is that it is arbitrary, because it is usually created by people. On the other hand, the phishing emails' body text disclose equivalent attributes, in deep semantics that differ from benign emails. Furthermore, phishing emails are usually comprised of multiple MIME parts (e.g., images and HTML code) that can be identified by the email's header fields, while benign emails mostly contain plain text.

All the emails undergo the parsing stage, wherein the corresponding body and header fields are extracted and stored in arrays B (equation (1)) and H (equation (2)) respectively, while the array L (equation (3)) stores the corresponding label of each email that represents the email's class. Where n is the number of samples (i.e., the number of both phishing and benign emails). The final step of the Email Parsing creates a joint array E (equation (4)) by merging the arrays B, H, and L. Array E is deployed by the content-based feature extraction stage, which extracts the content features from the body and header email fields, as well as by the pre-processing stage that prepares the emails' body texts for the text-based feature extraction stage.

$$B = [b_1\, b_2\, b_3 \ldots b_n] \qquad (1)$$

$$H = [h_1\; h_2\; h_3\; \ldots h_n] \qquad (2)$$

$$L = [l_1\; l_2\; l_3 \ldots l_n] \qquad (3)$$

$$E = \;[b_{1x1}\, h_{1x2}\, l_{1x3}\;\; b_{2x1}\, b_{3x1} \,\vdots\, b_{nx1} \quad h_{2x2}\, h_{2x3} \,\vdots\, h_{nx2} \quad l_{2x3}\, l_{3x3} \,\vdots\, l_{nx3}] \qquad (4)$$

3.3.2 Content-based Feature Extraction

Before elaborating on the content-based feature extraction procedure (S2 of Figure 3-1), the content-based features are analyzed, which are the first element of the proposed hybrid feature set. The deployment of the content-based features stems from the literature in the ML phishing email detection domain, where prior to the evolution of NLP methods, the existing works were heavily relied on the email's contents (i.e., features from specific parts of an email, such as headers, attachments, and suspicious words) to identify traits that accurately distinguish phishing from benign emails [16]. Thus, this thesis argues that the contents of emails include information that can be employed for phishing email detection. More specifically, the proposed methodology supports that phishing emails follow a particular format, where their contents differ from benign emails. Based on the format of emails as well as the phishing email traits, 4 feature categories have been identified:

- Phishing emails are designed to perform particular malicious actions (e.g., deceiving the victim to share personal information or click a malicious attachment). To accomplish these actions, phishing emails employ ***Body Features*** that are related to the body structure of emails.
- The text of phishing emails is comprised of strong persuasion traits that are represented by ***Syntactic Features*** from the emails' body and subject fields.
- The emails include a header field that generates ***Header Features***, which provide useful information about the email' type and the number of recipients.
- ***URL Features*** obtained from the hyperlinks and the domains of emails to indicate whether the email contains malicious hyperlinks or is generated by peculiar domains.

The content-based features are summarized in Table 5 and are further discussed below.

Body Features: The structure of phishing emails' body field differs from benign emails, mainly due to actions that an attacker does to deceive the victim to steal personal information or deliver malware. The body features category concentrates on traits that are located merely in the emails' body field and are depicted by binary and integer values. The binary values are represented by 0 or 1 and record whether the feature appears in an email, while integer values are represented by the number of appearances of a feature in an email.

1. HTML Code: This feature records the occurrence of HTML code in an email. (binary)
2. HTML Forms: This feature represents the presence of HTML-based forms in the email. HTML forms are frequently deployed by phishing campaigns to steal sensitive information. (binary)
3. Scripts: This feature tracks the presence of script code in the email (e.g. JavaScript). The scripts enhance the possibility for an email to include malicious code. The script codes are identified by the appearance of one of the following MIME types in the email: text/javascript, application/javascript, text/ecmascript, application/ecmascript. (binary)

4. Attachments: This feature represents the number of attachments in the email since phishing emails often deliver the malware via email attachments masqueraded as legitimate files (e.g., PDF). (integer)
5. Image Link: This feature records the presence of a hidden hyperlink behind an image. Phishing emails, frequently hide malicious hyperlinks (i.e., phishing URLs) behind images to draw the victim's attention and click the malicious link without their knowledge. (binary)

Syntactic Features: Overall, the syntax of benign emails is more abstract than phishing emails. While the syntax of different phishing email campaigns presents similarities since their purpose is to persuade, frighten, or deceive the receiver to perform certain actions (e.g., click on a hyperlink, share her credential, etc.). The syntactic features category aims to the identification of various linguistic traits of phishing emails that vary from benign emails. These traits are portrayed by binary and numerical values.

6. Bad words in email's body: This feature counts the occurrences of certain words in the email's body. Various words appear more frequently in phishing emails than others. These words are: "link", "click", "confirm", "user", "customer", "client", "suspend", "restrict", "verify", "payment", and "protect". (integer)
7. Bad words in email's subject: Similar to the feature Bad words in email's body various words that appear in phishing emails subjects. These words are: "verify", "bank", "debit", "payment", and "suspend". This feature preserves the number of occurrences of the aforementioned words in the email's subject. (integer)
8. Absence of "RE" in subject: This feature tracks the presence of "RE" at the beginning of the email subject. The "RE" abbreviation is used in the subject of emails to indicate that it is a reply to another email. Usually, phishing emails do not have this abbreviation. (binary)
9. No. of words in the email body: This feature preserves the total number of words that are included in the email's body. (integer)
10. No. of characters in the email body: This feature counts the total number of characters in the email's body. (integer)
11. Richness: This feature records the ratio of the total number of words to the total number of characters. (float)

$$Richness = \frac{No.of\ words\ in\ the\ email\ body}{No.of\ characters\ in\ the\ email\ body}$$

12. Total number of distinct words in the email body: This feature counts the total number of distinct words that are included in the email's body. (integer)
13. Number of email parts: Multipart emails combine in the email body field different data types e.g., images with HTML code). This integer feature includes the number of different parts in an email. (integer)

Header Features: The email headers when combined with features from the other categories can provide useful information for the detection of adversarial activity. This feature category is comprised of numerical and textual features.

14. Email encoding: This lexical feature records the email's encoding type using the Content-Transfer-Encoding header. The values of Email encoding feature are Base64, Quoted-Printable, 8Bit, 7Bit, Binary, X-token, and None. (textual)
15. Number of email recipients: This feature calculates the number of the email recipients. Often phishing campaigns simultaneously target multiple users. (integer)

URL Features: It is common for phishing emails to contain malicious URLs to be used as bait for the victims. To this end, a separate category of features that originated explicitly from URLs is considered. The URL features category is represented by binary and numerical values.

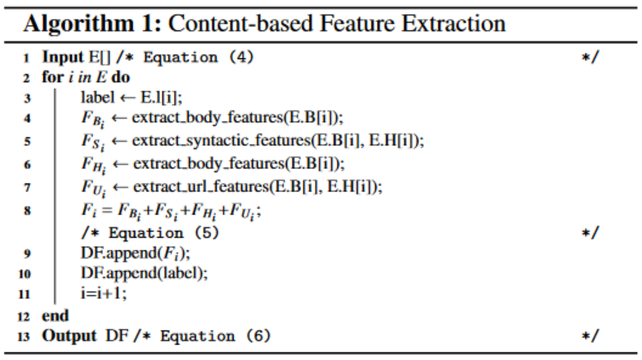
16. IP URL: This feature records the presence of a URL in an email that has an IP address instead of a domain name. It is common that temporary malicious URLs to be comprised of IP instead of domain names. (binary)
17. Number of hyperlinks: This feature calculates the number of hyperlinks that an email contains. (integer)
18. Text hyperlink: This feature represents the presence of human-readable text to hide a hyperlink. (binary)
19. Number of different HREF: This feature counts the cases in which the URLs of HREF HTML tags differ from the text that is appeared to the user. It has been noticed that phishing emails usually include hyperlink text which indicates a legitimate website, but instead they point to a phishing website. (integer)
20. Number of dots: This feature calculates the dots of each URL that appear in the email and records the highest number. It has been found that four dots or more is a usual indicator of a malicious URL [49]. (integer)
21. Port URL: This feature records the presence of URLs that contain a port number. Standard ports do not appear in the URLs and legitimate URLs rarely use non-standard ports. (binary)
22. Check domain: This feature tracks the presence of different sender's domain from the hyperlink's domain. (binary)

Please note that to the best of our knowledge, some of the features, such as *Absence of "RE" in the in subject, Email Encoding, Number of Email Recipients, Text Hyperlink, Check Domain*, have not been deployed by existing works that focused on the contents of emails; hence, the introduction of these features is part of the novelty of the proposed work. The content-based feature extraction stage extracts the above mentioned features and their numerical values from the email's contents. Thus, it is generated a joint set *F* that includes all the evident feature categories of the proposed feature set:

$$F = F_B \cup F_S \cup F_H \cup F_U \qquad (5)$$

Where the *body, syntactic, header,* and *URL* feature categories are represented by F_B, F_S, F_H, and F_U respectively. More specifically, utilizing the set F, it is defined an |F|-dimensional vector space that includes binary values, real numbers, integers, and lexical values. To be more precise, 9 features are binary values, 11 features are integer numbers, 1 feature is a real number (float), and 1 feature is represented by a lexical value. The value of each feature is shown in Table 3. The sets F of all the emails are grouped together into a DataFrame (DF) as depicted in Array *DF* (Equation 6). DF is a matrix in which the rows represent the emails, while the first columns represent the extracted content-based features and the last one depicts the label of the email (namely, 0 for benign and 1 for phishing). The labels of the emails are already known since the emails have been acquired from public email datasets. At this point, it should be clarified that the content-based features were extracted from the emails sequentially by a Python program, that receives as input the emails and generate the Array *DF* (equation (6)). Algorithm 1 represents the pseudocode of the content-based feature extraction stage. The input of the pseudocode (line 1) is the Equation 4 from the Email Parsing stage. Then, from each email (lines 2-12) the body, syntactic, header, and URL features are extracted (lines 4-8). Later, the Equation 6 is calculated based on the extracted features, which along with the label of the email (line 3) are used to construct Equation 6 that is the output of the pseudocode (line 13).

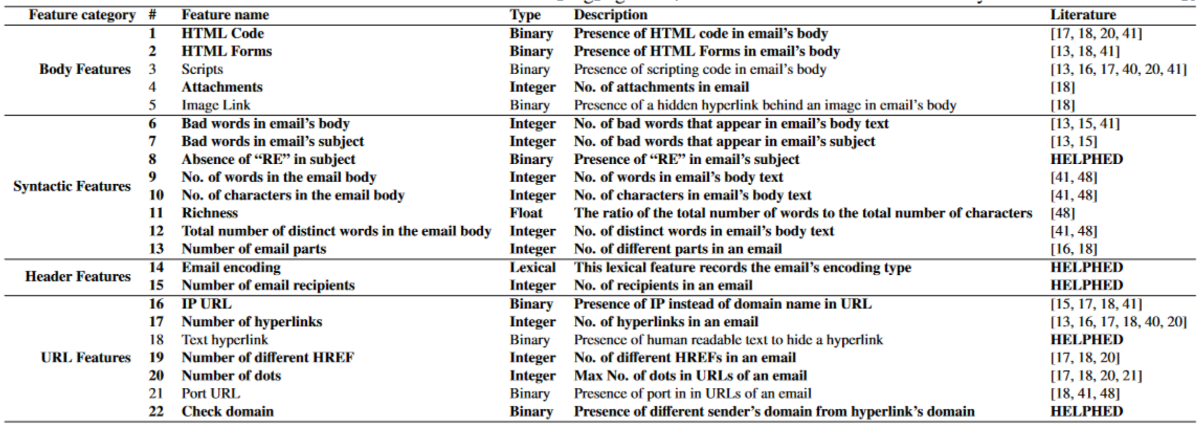
$$DF = \left[F_{B_1}\, F_{S_1}\;\; F_{H_1}\, F_{U_1}\, l_1\;\; F_{B_2}\; \vdots\; F_{B_n}\;\; F_{S_2}\; \vdots\; F_{S_n}\;\;\; F_{H_2}\; \vdots\; F_{H_n}\;\;\; F_{U_2}\; \vdots\; F_{U_n}\;\;\; l_2\; \vdots\; l_n \;\; \right] \qquad (6)$$



Algorithm 1: Content-based Feature Extraction

```
Input E[] /* Equation (4)                                   */
for i in E do
    label ← E.l[i];
    $F_{B_i}$ ← extract_body_features(E.B[i]);
    $F_{S_i}$ ← extract_syntactic_features(E.B[i], E.H[i]);
    $F_{H_i}$ ← extract_body_features(E.B[i]);
    $F_{U_i}$ ← extract_url_features(E.B[i], E.H[i]);
    $F_i = F_{B_i}+F_{S_i}+F_{H_i}+F_{U_i}$;
    /* Equation (5)                                         */
    DF.append($F_i$);
    DF.append(label);
    i=i+1;
end
Output DF /* Equation (6)                                   */
```

Table 5 Content-based Features. The HELPHED value highlights the features that have been introduced in this methodology



Feature category	#	Feature name	Type	Description	Literature
Body Features	1	HTML Code	Binary	Presence of HTML code in email's body	[17, 18, 20, 41]
	2	HTML Forms	Binary	Presence of HTML Forms in email's body	[13, 18, 41]
	3	Scripts	Binary	Presence of scripting code in email's body	[13, 16, 17, 40, 20, 41]
	4	Attachments	Integer	No. of attachments in email	[18]
	5	Image Link	Binary	Presence of a hidden hyperlink behind an image in email's body	[18]
Syntactic Features	6	Bad words in email's body	Integer	No. of bad words that appear in email's body text	[13, 15, 41]
	7	Bad words in email's subject	Integer	No. of bad words that appear in email's subject	[13, 15]
	8	Absence of "RE" in subject	Binary	Presence of "RE" in email's subject	HELPHED
	9	No. of words in the email body	Integer	No. of words in email's body text	[41, 48]
	10	No. of characters in the email body	Integer	No. of characters in email's body text	[41, 48]
	11	Richness	Float	The ratio of the total number of words to the total number of characters	[48]
	12	Total number of distinct words in the email body	Integer	No. of distinct words in email's body text	[41, 48]
	13	Number of email parts	Integer	No. of different parts in an email	[16, 18]
Header Features	14	Email encoding	Lexical	This lexical feature records the email's encoding type	HELPHED
	15	Number of email recipients	Integer	No. of recipients in an email	HELPHED
URL Features	16	IP URL	Binary	Presence of IP instead of domain name in URL	[15, 17, 18, 41]
	17	Number of hyperlinks	Integer	No. of hyperlinks in an email	[13, 16, 17, 18, 40, 20]
	18	Text hyperlink	Binary	Presence of human readable text to hide a hyperlink	HELPHED
	19	Number of different HREF	Integer	No. of different HREFs in an email	[17, 18, 20]
	20	Number of dots	Integer	Max No. of dots in URLs of an email	[17, 18, 20, 21]
	21	Port URL	Binary	Presence of port in in URLs of an email	[18, 41, 48]
	22	Check domain	Binary	Presence of different sender's domain from hyperlink's domain	HELPHED

3.3.3 Pre-processing

The pre-processing stage (S3 of Figure 3-1) takes as input the emails' body text (i.e., the first column of *E*) and aims the elimination of redundant and trivial information. Furthermore, it contributes to the creation of a consistent format for all the emails' text. The pre-processing of the text is a common procedure that has been applied in many text classification tasks and can be found in several existing works (e.g., [72]).

This stage aims to assist the textual feature extraction stage identifying the words of the email corpus that are more informative. The actions that pre-processing performs to accomplish its goals are depicted in Algorithm 2. The pseudocode has as input the body text of emails from Equation 4 of the Email Parsing stage (line 1). Then the pre-processing begins by separating the words of the email's body text (lines 4-15) and converting the words into lowercase (5-7). Next, it removes all the stop words, special characters, punctuation marks, and HTML elements (lines 8-10). The hyperlinks that may appear in the email's body text are replaced with a fixed string (lines 11-13). The reason behind using a fixed string and not a common word, like "URL" or "LINK", is that the common word may appear in the text, hence the results will be falsified. Later, a tokenization process is performed [23], where a delimiter is used to split the words to convert the text into a list of words. In this work, the delimiter is a white space (e.g., space, new-line, tab) (line 16). By converting the text into a list, it is easier to process the words and perform the final step of the pre-processing stage, which is the reduction of a word's inflectional forms (line 17). Finally, the output is the pre-processed body text of the emails (line 18).

Algorithm 2: Email Body Text Pre-processing

```
Input E[];
/* Equation (4)                                      */
e ← E.b[];
e ← e.split();
for w in e do
    if w is uppercase then
        convert w to lowercase;
    end
    if w=special_character OR w=stop_word OR w=HTML_element then
        remove w;
    end
    if w is hyperlink then
        w←"LINKLINK";
    end
    w_new ← w;
end
w_new ← w_new.split(" ");
w_new ← w_new.lemmatization();
Output Pre-processed Text;
```

Lemmatization and stemming [129] are the two most well-known processes for the reduction of a word’s inflectional forms to convert it in its root form resulting in a smaller word set, since all the inflections of a word are converted to one (e.g., the words “better”, “best”, and “good” have the same root). Although they share the same purpose, stemming and lemmatization heavily differ. Stemming only trims the end or/and the beginning of a word based on prefixes and suffixes of inflected words. On the other hand, lemmatization is based on a thorough morphological analysis of the words and hence provides a meaningful root form of the words. To do so, the lemmatization process requires a vocabulary that contributes to the identification of the words’ lemma (i.e., the word’s original form). For the purposes of this work, the lemmatization method was deployed, since it provides a more accurate depiction of the words that appear in the emails and the WordNet database [130] was used as a vocabulary. WordNet is a comprehensive lexical repository mainly of the English language that includes semantic relations between words, which also extends to more than 200 languages. In WordNet, the words are linked into semantic relations and grouped into cognitive synonym sets (synsets) that express the meaning of a notion. The interconnection among the synsets provides a mesh of substantially related words and notions that enhance the outcome of the NLP task.

In this work, WordNet aims to the reduction of the semantics of the emails’ texts. Namely, via the lemmatization process, the diversity of the text is decreased by changing a group of words that have similar synonyms with their lemma. For example, the lemma of the word “sender” is the word “send”, thus all the words “sender” in the email dataset are replaced with the word “send”. Furthermore, the Part-of-Speech (POS) tagging method has been utilized to identify the grammatical category of each word (i.e., noun, verb, adjective, and adverb). POS helps the lemmatization process to further decrease the word instances. As one can understand, the lemmatization process significantly contributes to the dimensionality reduction of the emails’ body text corpus, namely it facilitates the textual feature extraction stage to identify and extract more meaningful features.

3.3.4 Text-based Feature Extraction

This section elaborates on the textual feature extraction stage (S4 of Figure 3-1), providing details about the procedures that followed to result in the text-based feature set (i.e., the second element of the hybrid feature set). Latest works deal the phishing email detection as a text classification task exploiting the NLP advances in the phishing email detection domain and avoiding the manual extraction of features. To achieve the best classification results, comprehensive and adequate information about the data (i.e., emails) is needed. Thus, the methodology considers the representation of emails as an integration of content-based and text-based features. Since emails’ body text is controlled by people, it is arbitrary. On the contrary, phishing emails have a specific purpose, which is to obtain sensitive information from the victims, hence the body text message is often comprised of a warning, a threat or a fascinating content that aims to exploit people’s psychological weaknesses and manipulate them to perform certain actions (e.g., click on a link). This renders the phishing emails’ body text to be different in deep semantics than benign ones. Numerous NLP methods have been proposed in the literature to extract text-based features from phishing emails that are deployed by ML algorithms to distinguish benign and phishing emails, however, the two most used are the TF-IDF [131] and Word2Vec [75].

A challenging task of this stage is to identify the method that can effectively process the emails' body text to convert it in a form that can be used as input to ML/DL algorithms. To this end, three well-known methods were compared, named TF-IDF, Word2Vec, and BERT. More specifically, these methods compared under the same settings (i.e., same dataset, experimental environment, ML algorithms) and evaluated in combination with ML algorithms to identify which pair (text-based feature extraction method-ML algorithm) can lead to better classification results. Five ML algorithms was deployed to parse the output of the text-based feature extraction methods and two datasets (one balanced and one imbalanced). The experiments concluded that the Word2Vec method is the most appropriate for an ML-based phishing email detection approach that focuses on the emails' body text, since Word2Vec outperformed both TF-IDF and BERT in all the experiments.

Based on the above mentioned, to effectively identify the deep semantics of emails, in this stage of the methodology, the text-based features are extracted from the emails using the Word2Vec method. The output of the pre-processing stage is employed by the Word2Vec method that converts the email's body text into text-based features without manual interference. The rest of the subsection elaborates more on the Word2Vec method to assist the reader understand the textual feature extraction process. Word2Vec [75] advances the text classification task and offers an individual perspective to the text mining domain due to its ability to convert words and phrases into vector representations. Word2Vec is a word embedding method that is able to capture the linguistic context of a word in a text, namely it identifies the relation of a target word with the surrounding words (e.g., the semantic and syntactic similarity). A two-layer ANN is employed to find the words' linguistic context, where the input is a large corpus of texts, which in this work is a large collection of emails and the generated outputs are vectors of particular dimensions. Specifically, Word2Vec creates a vectorized representation of words, named embeddings, in which each unique word in the collection has a corresponding vector linked with it and reflects the context of the word. Furthermore, similar words have similar vector representations, namely, they are close distance-wise in the embedded space. Word2Vec supports two techniques, (a) Continuous Bag of Words (CBOW), and (b) skip-gram. The CBOW technique predicts the target word-based on the distributed representations of the context (e.g., surrounding words), while skip-gram technique parses the target word to identify the surrounding words. The CBOW method is faster than skip-gram but skip-gram works better with infrequent words. Thus, in the methodology, the skip-gram method was utilized together with the hierarchical softmax method for the training of the model. The train complexity of skip-gram is calculated by:

$$Q = C * (D + D * log_2(V)) \qquad (7)$$

where C is the maximum distance between the words, D is the word representation, and V is the dimensionality. The probability of correctly predicting a word w_i by giving a word w j using the softmax method is depicted below. The two vectors u_w and v_w represent the word and the context respectively, since in skip-gram every word w is connected with the aforesaid vectors.

$$p(w_i|w_j) = \frac{e^{u_{w_i} v_{w_j}}}{\sum_{i=1}^{V} \; e^{u_i v_{w_j}}} \qquad (8)$$

In total 300 text-based features have been extracted from the emails using the above mentioned equations (7) and (8).

3.3.5 Feature Selection

The feature selection stage (S5 of Figure 3-1) is an essential data processing step that identifies the most informative features from the original feature set to enhance the performance of the ML classifiers in terms of accuracy and training time [132]. Its main objective is to remove irrelevant or redundant features without losing critical information. The feature selection process aids the methodology to train the algorithms faster, avoiding overfitting, and obtaining better prediction results. Therefore, the features are ranked by a utility measure, through which those with the greatest values will be selected based on a threshold value. The remaining (unselected) features, namely those with values less than the threshold value, are deleted and are not used by the ensemble classification stage.

Overall, the feature selection methods can be classified into four categories: filter, wrapper, embedded, and hybrid. The filter-based methods do not rely on a learning algorithm to identify the most informative features, instead, they evaluate the features according to heuristics based on general characteristics of the training data. On the other hand, the wrapper, embedded, and hybrid methods rely on a ML algorithm to identify the most informative features. More specifically, these methods conclude to the most informative features based on the algorithm's performance. Namely, they identify the most informative features specifically for the deployed algorithm. Hence, the latter methods contradict the generalization scope of the feature selection stage (i.e., to identify the most informative features for all the ML/DL algorithms). Moreover, another well-known method that has been used before for feature selection is PCA [133]. In order to understand why the PCA method was not deployed, it should be clarified that the main goal of the methodology's feature selection stage is to identify the most informative content-based and text-based features to enhance the training and classification performance of the Ensemble Learning methods. The PCA method is not a feature selection method per se, namely, it does not select the most informative features from a feature set; instead, it creates a new feature set with fewer features based on the original feature set [134]. Hence, in the literature is considered a feature extraction method. Existing works in the phishing email detection domain that deploy the PCA method, such as [17] and [70]; also support the argument that it is a feature extraction method rather than a feature selection method. Therefore, the PCA method is inappropriate for the feature selection stage of this methodology. Based on the above observations, it is deployed a filter-based method, named Mutual Information (MI).

MI is a well-known method for calculating the mutual dependence between two variables (or features), according to the entropy of a random variable. Particularly, MI calculates the quantity of information that is obtained in a random variable by observing another random variable, namely identifying the quantity of information each feature offers to determine the email's class. For two jointly continuous random variables, the MI is calculated by the equation (9):

$$I(X,Y) = \int_y \quad \int_x \quad p_{(X,Y)}(x,y)\ log\ log\ \frac{p_{(X,Y)}(x,y)}{p_X(x)p_y(y)} d_x\ d_y \qquad (9)$$

where *I(X, Y)* is a measure of dependency between the density of variable *X* and the density of the target *Y*, $p_x(x)$ and $p_y(y)$ are the probability densities of *x* and *y* respectively, and $p_{(X,Y)}(x, y)$ is the probability of joint density [58]. Finally, the input data in this task is the hybrid features, namely the content and text-based feature sets. Respectfully to the MI, the equation (9) was deployed to calculate the value of each feature, and based on these values the most informative features were identified that will be used in the classification stage. The feature selection process is a challenging task due to the hybrid features. Thus, in order to obtain more robust features, we processed the content-based and text-based feature sets separately. Particularly, using the equation (9) it was estimated the MI score for each feature and classify them in descending order. The feature selection stage concluded in 18 content-based features (presented with bold font in Table 3) and in 253 text-based features. Thus, the final hybrid feature set contains 271 features.

3.3.6 Ensemble Classification

The Ensemble Classification stage (S6 of Figure 3-1) is the nucleus of this methodology. Since, the email samples (i.e., emails) are classified as phishing or benign, the Ensemble Classification stage is a binary classification task. The deployed email samples were obtained from publicly available sources, viz the class of the emails is already known. Hence, the ensemble learning procedure of this stage is based on supervised learning.

The objective of the Ensemble Classification stage is to effectively process the hybrid features enhancing the classification performance. Towards this direction, instead of using a single classifier, this methodology is based on Ensemble Learning. Ensemble Learning combines the results of multiple base learners (i.e., ML algorithms) to achieve more stable, robust, and accurate prediction performance than could be obtained by any of the base learners alone. In Ensemble Learning, the prediction results outputted by base learners are further processed (e.g., by a fusion model) to generate the final classification prediction. To augment the efficacy of the methodology, two well-known Ensemble Learning methods were employed, named Stacking Ensemble Learning (Method 1) and Soft Voting Ensemble Learning (Method 2). These methods were chosen because they can include different ML algorithm types, namely, they offer the flexibility to employ various ML algorithms to process the hybrid features separately (i.e., using an ML algorithm to process the content-based features and a different ML algorithm to process the text-based features). Method 1 and Method 2 differ in the way they fuse the results of the base learners and conclude to the final classification result.

An innovation of this methodology is that the Ensemble Learning methods, which are employed to effectively process the hybrid features, are comprised of two base learners. More specifically, the hybrid feature set includes the content-based and text-based feature subsets. Each base learner processes these subsets separately, yet in parallel (e.g., the 1st base learner processes the content-based features, while the 2nd base learner processes the text-based features). The benefit of this approach is that it can deploy as base learners the ML algorithms that attain the best classification performance with each feature set.

Later, the results of the base learners are parsed by the fusion model to conclude to the final classification prediction. Moreover, the selection of the base learners that will be employed in Method 1 and Method 2 is a challenging task, since there are numerous ML algorithms appropriate for binary classification. To limit the available options the methodology focused only on ML/DL algorithms that have been previously applied in phishing email detection [135] accomplishing promising performance. Specifically, a pool of ML algorithms was created that included: 1) the function-based Logistic Regression (LR), 2) the probability-based Gaussian Naive Bayes (GNB), 3) the ANN Multilayer Perceptron (MLP), 4) the instance-based k-Nearest Neighbors (KNN), and 5) the tree-based Decision Tree (DT). Based on a thorough evaluation detailed in the next section, it was concluded that DT and KNN are the best base learners to process the content-based and text-based features respectively. The rest of the section elaborates on Stacking Ensemble Learning and Soft Voting Ensemble Learning.

3.3.6.1 Method 1: Stacking Ensemble Learning

In the methodology presented in this thesis, the Stacking Ensemble Learning (Method 1) [86] (depicted as *Method 1 - Stacking Classification* in Figure 3-1) reinforces the detection of phishing emails by exploiting the predictions of two individual base learners, which process the hybrid features separately and in parallel, as well as by utilizing a fusion model (in Stacking Ensemble Learning the fusion is performed by an ML/DL algorithm) that combines these predictions to accurately classify the emails between phishing and benign. As depicted in Figure 4-2, the Stacking Ensemble Learning model is comprised of 2-tiers. Tier-1 performs an initial classification of the emails, where the base learners classify them using the derived hybrid features, namely the content-based and the text-based feature sets. Particularly, the DT algorithm classifies an email based on the email's content-based features, while the KNN classifies an email based on its text-based features (Tier-1 of Figure 3-2). The output of Tier-1 (i.e., the classification result of DT and KNN) is parsed in Tier-2, where a fusion model is applied. In Method 1, an ML algorithm is applied as a fusion model to perform the final classification (Tier-2 of Figure 3-2). Based on the evaluation explained in Section 3.4, the MLP DL algorithm was deployed as the fusion model since it can accomplish high detection performance with both the content-based and text-based features. Particularly, the MLP classifier parses the predictions of the DT and KNN base learners for each email, which is 1 for phishing and 0 for benign, and decides whether an email is phishing or benign.

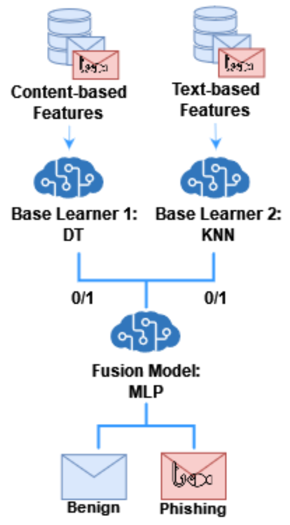


Figure 3-2 Stacking Ensemble Classification (Method 1)

3.3.6.2 Method 2: Soft Voting Ensemble Learning

The Soft Voting Ensemble Learning (Method 2) (depicted as Soft Voting Classification in Figure 3-1) considers the class prediction probabilities of each base learner and combines these predictions through an average function to perform the final classification. In general, there are two voting Ensemble Learning methods: a) Majority Voting [133] and b) Soft Voting [134], which differ on the way they process the results in the fusion model. Particularly, the Majority Voting takes the output of the base learners and performs the final classification based on the majority votes, while Soft Voting, averages the decisions of each base learner to perform the final classification. In this way, when compared with Majority Voting, the Soft Voting method eliminates the structure sensitivity that occurs from the base learners, reduces the ensemble's variance, and obtains more information.

As presented in Figure 3-3, correspondingly to Method 1, two base learners are also employed in Method 2, namely the DT algorithm that processes the content-based features and the KNN algorithm that processes the text-based features (Tier-1 of Figure 3-3). The output of both base learners is the probability that an email belongs to a certain class (e.g., *P(phishing)* or *P(benign)*). In Tier-2, these probabilities are used as input to an Argmax function to identify the most probable class and perform the final classification. More specifically, phishing email detection is a binary classification problem, where $Y=0,1$ (0 for benign and 1 for phishing) and a feature space $X \in \Re^k$, such that for any base learner F holds the function: $X \rightarrow Y$, then for each email sample (i.e., email) the decision profile of a base learner i is a pair of class probabilities $[P_{i_0}, P_{i_0}]$, where$\sum \left((P_{i_0}, P_{i_1})\right) = 1$. Consequently, the fusion model of the Soft Voting Ensemble Learning method, given an email sample X_m, combines the probabilities of the base learners to identify the most probable class *(c)* using the equation (10).

$$y_m = argmax \sum_{i=1}^{|p|} P_i(c|x_m), y_m \in Y, m \in \{1,2,\ldots,n\}, \qquad (10)$$

where |p| is the number of the base learners.

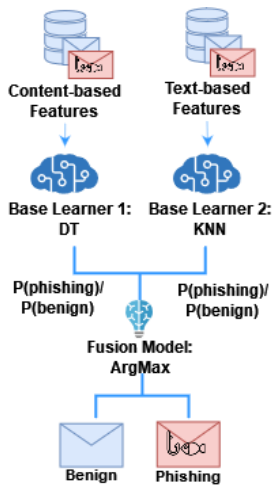


Figure 3-3 Soft-Voting Ensemble Learning (Method 2)

3.4 Experimental Evaluation

In this section, it is presented a detailed evaluation of the proposed methodology focusing both on the deployed data and the prediction results. The experiments conducted in the chapter of this thesis were performed in a virtual machine located in a VMWare ESXi server with Intel Xeon 4114, 2.20 GHz x 4 CPU, and 12 GB RAM. The OS of the virtual machine was Ubuntu 20.04 64-bit. The proposed methodology was developed using Python Scikit-learn[26] and Apache Spark[27]. The Scikit-learn library was selected because it supports various state-of-the-art ML and DL algorithms that can be easily configured, resulting in designing appropriate models (e.g., by defining the values of hyperparameters) based on our needs (i.e., efficiently process the emails) and our goal (i.e., to enhance the phishing email detection performance). The evaluation approach along with the deployed evaluation rules that assist in avoiding biased results are described in Section 3.4.1. Section 3.4.2 elaborates on the datasets employed to evaluate the proposed methodology. In section 3.4.3, the utility metrics to assess the outcomes of the proposed methodology are presented. Section 3.4.4 discusses the feature evaluation process, while section 3.4.5 analyzes the procedure that is followed to select the best base learners for the ensemble classification stage comparing their performance on content-based and text-based features separately. The evaluation results are described in section 3.4.6 and finally, section 3.4.7 emphasizes on a discussion between the proposed methodology and existing works. It should be noted that the results cannot be thoroughly explained, since the deployed hybrid features are not interpretable (because the text-based features are vector representations of the words that appear in the email); however, the results are explicated based on the traits of the proposed methodology (e.g., ML/DL algorithms).

3.4.1 Evaluation Approach

Several existing works, such as [32], [16], [136], and [137], have identified inaccuracies and pitfalls in the assessment of ML-based detection solutions, undermining the veracity of their obtained experimental results. These inaccuracies have been also identified in the phishing email detection domain [15]. Thus, to comprehensively and properly evaluate the detection capabilities of the proposed methodology, a set of state-of-the-art rules for the evaluation of ML algorithms in the phishing email detection field were introduced. In the following, the most relevant rules are presented that were taken into account to increase the robustness of the proposed methodology and avoid attaining misleading results.

[26] https://scikit-learn.org/stable/
[27] https://spark.apache.org/

1. R1: Several existing works (e.g., [78]) share limited information regarding the datasets, the experimental environment, the features, and the ML algorithms, hence it is not possible to reproduce their implementation and repeat the experiments. Researchers should pay more attention to the reproducibility of their work.
2. R2: The deployed phishing email samples should consider the evolution of phishing email attacks. Several recent works, such as [83], utilize old phishing emails (before 2015) that do not reflect the current attack trends. Violation of this rule means that obsolete phishing emails are deployed, and numerical results may be inflated.
3. R3: The majority of existing works have been evaluated in balanced datasets or almost balanced (e.g., [61], [69], [79]), namely the number of benign and phishing samples are almost the same. However, in real life, phishing email detection is an imbalanced classification problem, since the number of benign emails that an organization receives is much higher than phishing ones, hence in order to represent a realistic scenario the assessment should be performed on an imbalanced class ratio.
4. R4: The ML-based detection method should be evaluated using appropriate evaluation metrics. Even though this may seem evident, there are plenty of works in the literature, such as [71] that use inappropriate evaluation metrics, namely they evaluate their method on an imbalanced class ratio by employing metrics that are influenced by class imbalance (e.g., detection accuracy).
5. R5: A significant body of existing works did not employ diverse data sources, hence they lack generalization (e.g., [72]). For instance, when using benign emails only from a specific organization the model learns to distinguish the particular email format that represents the organization's needs and if applied to another organization in a different field it might lead to more false positives.

The proposed work complies with all the above rules. More specifically, regarding *Rule R1*, the source code of the experiments and the evaluation dataset have been publicly shared along with [31]. The next section elaborates on the dataset and presents how the assessment complies with the remaining rules (i.e., R2-R5).

3.4.2 Dataset

Particular attention has been given on the evaluation of the proposed methodology; therefore, a realistic dataset was created that contains samples from various publicly available sources, such as the Enron Email Corpus [62], the SpamAssassin Public Corpus [73], the Nazario Phishing Corpus [60], and authors' mailboxes (compliance with Rule R5). Before elaborating on each corpus, it is important to mention that the IWSPA dataset [138], which is a well-known dataset among existing works, was not utilized because the dataset's creators have removed information that is important for the content-based features (e.g., headers, domains, and IP addresses), hence it could not be used in our experiments.

The Enron Corpus is a large dataset that is comprised of approximately 500,000 emails originated from 158 employees of the Enron Corporation. It is one of the few publicly available mass collections of real emails and it has been used in the past in various researches [74], [139], [140]. The emails that correspond to interactions among different employees on day-to-day work issues, were publicly released during the legal investigation of the corporation. From the Enron Corpus, 28,000 emails were randomly selected to comprise the benign email samples, which is the highest number among existing works. Only a part of the Enron Corpus emails was deployed since in case the whole corpus was used, the benign samples of our dataset would not be diverse. Moreover, Spamassassin Corpus contains real email messages that originated from the Spamassassin developers, and it was originally created for testing the Spamassassin filter. It contains in total 6,047 emails of which 1,897 are spam and the remaining 4,150 are benign. The benign emails are further categorized on 3,900 easy ham (easy to differentiate them from spam emails), and 250 hard ham (difficult to differentiate them from spam emails). In order to increase the variety of the benign samples, all the benign emails from the Spamassassin Corpus (both easy ham and hard ham) were included. After merging the Enron and Spamassassin Corpuses and dropping the duplicates the total benign email samples were 32,046.

The Nazario Phishing Corpus has been used as the source of raw phishing email data, which is also a public dataset that contains more than 4,000 phishing emails approximately from 2003 to 2020 (at the time of writing) and it has been widely used to validate relevant research works. To consider the evolution of phishing emails, all the phishing emails from 2015-2020 (namely, 1,472 emails) and 1,992 emails that originated before 2015 have been deployed in order to create a realistic scenario, where the phishing emails samples are approximately 10% of all the benign samples (the number of benign samples as mentioned above is 32,046). Furthermore, 76 phishing emails from the authors' mailboxes have been employed to enhance the diversity of the phishing samples as well as to include more recent phishing emails, since these emails have been received in the last couple of years (e.g., 2019-2021). The final phishing email dataset contains 3,465 samples.

A common drawback of several existing works is that their assessment did not consider the evolution of phishing email attacks. Therefore, to abide by Rule R2 1,548 phishing emails from the period 2015 to 2021 were utilized. Moreover, to comply with Rule R3, an imbalanced class ratio has been deployed. Particularly, 1:10 ratio has been used between phishing and benign email classes (namely, the phishing emails comprise approximately 10% of the benign samples). The dataset was split into training and testing subsets, where 70% of the samples were used for training/validation and the remaining 30% used for testing the trained models, namely the training and testing sets are comprised of 24,857 (22,432 benign and 2,425 phishing) and 10,654 (9,619 benign and 1,035 phishing) samples respectively. Table 6 depicts the number of email samples that constitute the training and testing sets as well as the total number of samples. Finally, to the best of our knowledge, the deployed dataset is the largest among recent existing works. The rationale for employing one large dataset instead of multiple smaller ones lies on the fact of representing a more realistic scenario (i.e., the emails originated from different organizations and the benign emails are much more than phishing). In this way, it is also addressed a limitation of the literature that identified by [15], which highlights the fact that existing solutions have been tested on smaller datasets that usually include samples originated from one source, hence their models might learn to distinguish the particular email format that represents the organization's needs and if applied to another organization in a different field it might lead to more false positives.

Table 6 The details of the evaluation database

Dataset	**Benign**	**Phishing**	**Total**
Training set	22,432	2,425	24,857
Testing set	9,619	1,035	10,654
Total	32,051	3,460	35,511

3.4.3 Metrics

The experimental results of the proposed methodology were based on several well-known metrics that offer a clear view of its prediction capabilities (compliance with Rule R4). The True Positive (TP) is the number of phishing emails that have been classified correctly and the True Negative (TN) is the number of benign emails that have been classified correctly. The False Positive (FP) is the number of benign emails that have been classified erroneously and the False Negative (FN) is the number of phishing emails that have been classified erroneously. The evaluation of the proposed methodology relied in the following metrics:

- Recall: Illustrates the correctly classified phishing emails compared to the total number of phishing emails:

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

- Precision: Represents the correctly classified phishing emails compared to number of emails that classified as phishing:

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

- False Positive Rate (FPR): Depicts the mistakenly classified benign emails compared to the total number of benign emails:

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

- False Negative Rate (FNR): Refers to the mistakenly classified phishing emails compared to the total number of phishing emails:

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

- Accuracy: The percentage of correctly classified emails compared to the total number of emails:

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

- F1-Score: A measurement to assess the prediction accuracy that considers both the precision and recall. The F1-score shows a better metric in case of imbalance datasets:

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

- Matthews Correlation Coefficient (MCC): Is a measurement to assess binary classifications that is appropriate for imbalanced datasets, because it examines the accuracies and error rates on both classes [141]. The higher the correlation between “true” and predicted values, the better the prediction.

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

- Receiver Operating Characteristic (ROC) Curve: Is a graph that plots the TPR and the FPR. More specifically, it illustrates the performance of a classification algorithm at different classification thresholds. Please note that the closer the ROC curve is to the shape of an upside down L, the better.
- Area Under the ROC Curve (AUC): Measures the area under the ROC curve.
- Training time: The time that is required for a ML model to be trained is an important factor that most of the existing works neglect. Thus, the training time of a model was measured in seconds.

3.4.4 Feature Evaluation

This section elaborates on the evaluation of the content-based and text-based features and describes how the final hybrid feature set was calculated using the MI feature selection method that described in Section 3.3.5. Figure 3-4 depicts a diagram of the content-based features and their MI scores. The MI scores of the content-based features ranged between 0.20912 and 0.001813. In order to identify the most informative features for the detection of phishing emails, various experiment have been performed, where each time a different feature set was used based on the features’ MI scores. More specifically, we picked: i) features with above 0.1 MI score (4 features), ii) features with above 0.05 MI score (11 features), iii) features with above 0.01 MI score (18 features), and iv) features with above 0.001 MI score (all 22 features). Overall, all the algorithms accomplished the best detection performance when selecting the feature set with above 0.01 MI score that included 18 features. Please note that the results of MLP were similar when employing MI scores of above 0.01 and 0.001, namely with 18 and 22 features; hence to achieve the best results with the least possible overhead, the feature selection stage led to 18 content-based features. Table 7 shows the best 5 content-based features based on their MI scores.

Regarding the text-based features, Figure 3-5 illustrates a diagram of the text-based features and their MI scores. To identify the most informative features, the same approach was utilized (i.e., the features' MI scores were estimated and classified in descending order.) but employed more MI score thresholds to create more feature sets, since the text-based feature set (300 features) is significantly bigger than the content-based (22 features). Particularly, the MI scores of the text-based features ranged between 0.1439 and 0.00228. As previously mentioned, several experiments have been performed using different features each time. The deployed features were: i) features with above 0.1 MI score (14 features), ii) features with above 0.05 MI (93 features), iii) features with above 0.02 MI score (190 features), iv) features with above 0.015 MI score (217), v) features with above 0.01 MI score (253), vi) features with above 0.002286 MI score (300). The results when employing 190, 217, 253, and 300 features were close for all the ML algorithms, however, the best detection performance was attained when employing 253 features. Thus, the feature selection stage resulted in 253 text-based features. The 5 best text-based features are depicted in Table 8.

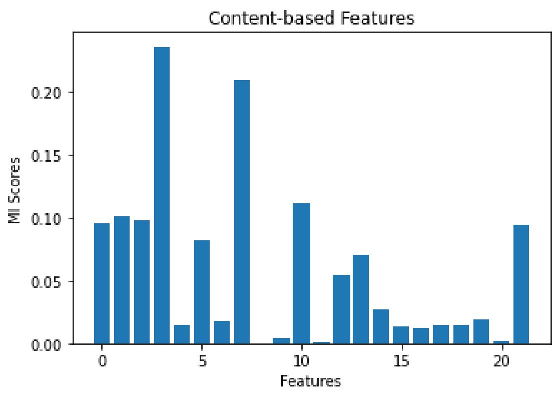


Figure 3-4 Content-based Features

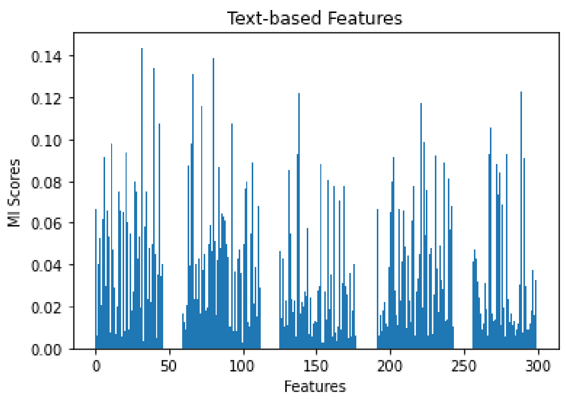


Figure 3-5 Text-based Features

Table 7 Top 5 Content-based Features

Feature	**MI Score**
HTML code	0.2361
Number of different HREF	0.2096
Number of dots	0.1112
Email Encoding	0.1018
Number of email parts	0.0978

Table 8 Top 5 Text-based Features

Feature	**MI Score**
Feature 32	0.1439
Feature 80	0.1391
Feature 40	0.1341
Feature 250	0.1331
Feature 66	0.1312

3.4.5 Selection of Base Learners

The selection of the two ML/DL algorithms (i.e., one algorithm will focus on the content-based features, while the other will focus on the text-based features) that will be deployed as base learners is a challenging task of the Ensemble Learning stage. This task is focused on the evaluation of the ML/DL algorithms on each feature set separately to identify the most efficient algorithm on each feature set. More specifically, several well-known ML (LR, GNB, KNN, DT) algorithms were evaluated and one DL algorithm (MLP) on content-based and text-based feature sets separately, using the datasets presented in Table 4, and the evaluation metrics discussed in Section 3.4.3. MLP was tested using various hidden layers (1, 2, 3, 4) and concluded that it accomplishes the best performance with 3 hidden layers. Following, it was demonstrated why HELPHED (both in Method 1 and Method 2) deploys the DT and KNN ML algorithms as base learners as well as why MLP was used as a fusion algorithm in Method 1.

Regarding the results on the content-based features, the DT algorithm revealed the best classification performance on all the evaluation metrics (except training time), reaching 0.9885 F1-score, accuracy, precision, and recall, 0.9677 AUC, 0.9615 MCC outperforming all the other ML/DL algorithms, as presented in Table 9. More specifically, DT concluded to 9,599 TNs, 42 FNs, 20 FPs, and 993 TPs. It is worth mentioning that the second-best performance achieved by the MLP algorithm (0.9806 F1-score, accuracy, and recall, 0.9809 precision, 0.9517 AUC, and 0.8909 MCC) as well as that the worst performance attained by the KNN algorithm (0.9387 F1-score, 0.9445 accuracy, 0.9406 precision, 0.9445 recall, 0.7589 AUC, and 0.6406 MCC). Based on the evaluation results on the content-based features, DT is the most appropriate algorithm for the classification of emails using only the content-based features. Thus, DT was deployed as the first base learner that processes the content-based features in HELPHED.

Regarding the results on the text-based feature set, KNN and MLP algorithms achieved comparable performance, as depicted in Table 10; however, the performance of KNN was slightly better. Particularly, KNN attained 0.9862 F1-score, accuracy, and recall, 0.9863 precision, 0.9635 AUC, 0.9218 MCC, as well as a very low training time (0.005 sec) as a result of 9,539 TNs, 67 FNs, 80 FPs, and 968 TPs. While, MLP accomplished 0.9812 F1-score, 0.9818 accuracy and recall, 0.9816 precision, 0.9179 AUC, and 0.8923 MCC which were the result of 9,592 TNs, 167 FNs, 27 FPs, and 868 TPs. Contrariwise, the GNB algorithm revealed the worst performance (0.9369 F1-score, 0.932 accuracy and recall, 0.9466 precision, 0.8955 AUC, and 0.6831 MCC). Based on the evaluation results on the text-based features, KNN is the most appropriate algorithm for the classification of emails using only text-based features. Therefore, KNN was employed as the second base learner that processes the text-based features in HELPHED.

At this point, to encourage the reader to understand the presented notions, the possible reasons were provided (based on the traits of the deployed ML algorithms) that led to the best results. When it comes to the results using the content-based feature set, one possible reason that DT outperformed all the other algorithms is because there are features that are more informative than others and DT relies on them to perform the classification more than the other algorithms. Based on the features' evaluation (see section 3.4.4), the 3 most informative features that facilitate DT reveal the best performance are the HTML code, number of different HREF, and number of dots. Considering the results using the text-based features, the potential reason that KNN attained better detection performance than the other algorithms is due to its ability to process a high number of features, namely the 252 text-based features, more effectively than the other algorithms. While MLP achieved comparable performance with KNN because a DL algorithm can deal both with a high number of samples and features more efficiently than the ML algorithms. Moreover, MLP accomplished very good performance (more than 98%) with both content-based and text-based features, thus it was deployed as a fusion algorithm in Method 1.

Table 9 Evaluation on content-based features

Classifier	**F1-score**	**Accuracy**	**Precision**	**Recall**	**AUC**	**MCC**	**Training time**	**TP**	**TN**	**FP**	**FN**
LR	0.9756	0.9747	0.9778	0.9747	0.9674	0.8697	1.21	9392	992	43	227
GNB	0.972	0.9711	0.9737	0.9711	0.9499	0.8476	0.006	9390	956	79	229
KNN	0.9387	0.9445	0.9406	0.9445	0.7589	0.6406	0.003	9516	547	488	103
DT	0.9885	0.9885	0.9885	0.9885	0.9677	0.9615	0.016	9599	993	42	20
MLP	0.980	0.9806	0.9809	0.9806	0.9517	0.8909	3.64	9499	948	87	120

	6										

Table 10 Evaluation on text-based features

Classifier	**F1-score**	**Accuracy**	**Precision**	**Recall**	**AUC**	**MCC**	**Training time**	**TP**	**TN**	**FP**	**FN**
LR	0.9706	0.9692	0.9742	0.9692	0.9644	0.8147	64.467	9334	**992**	**43**	285
GNB	0.9369	0.932	0.9466	0.932	0.8955	0.6831	0.078	9049	880	155	570
KNN	**0.9862**	**0.9862**	**0.9863**	**0.9862**	**0.9635**	**0.9218**	**0.005**	9539	968	67	80
DT	0.9672	0.9671	0.9673	0.9671	0.9102	0.8163	0.284	9434	869	166	185
MLP	0.9812	0.9818	0.9816	0.9818	0.9179	0.8923	13.093	**9592**	868	167	**27**

3.4.6 Experimental Evaluation

This section presents the evaluation of the proposed methodology (both with Method 1 and Method 2) on classifying the emails using the hybrid features. To highlight its effectiveness (i.e., the integration of Ensemble Learning with hybrid phishing email features), it was compared with traditional (state-of-the-art) ML and DL classifiers (i.e., LR, GNB, KNN, DT, and MLP). Furthermore, proposed methodology was also compared with a prominent Ensemble Learning method, named Random Forest (RF) [69] to justify the advantage of the proposed methodology that processes the hybrid features separately (i.e., using an ML algorithm to process the content-based features and another ML algorithm to process the text-based features) against RF that processes the hybrid features like a unified feature set. The experiments were performed based on the evaluation guidelines described in Section 3.4.1 to avoid misleading or biased results. The dataset presented in Section 3.4.2 was deployed. The results revealed in this section were based on the metrics that were analyzed in Section 3.4.3, acquired from both email classes, namely phishing and benign, and their respective samples in the testing set (i.e., 9,619 benign and 1,035 phishing).

The results arranged in Table 11 present the performance of the ML classifiers as well as the performance of the proposed methodology with Method 1 and Method 2. As shown in Table 9, LR and GNB were poorly performed, hence their results will not be further discussed. Overall, the classification results of the remaining algorithms are enhanced when the hybrid features are employed (i.e., when considering both the content and text of emails). Particularly, in comparison to ML and ensemble RF classifiers, both Method 1 (Stacking) and Method 2 (Soft Voting) improved the overall detection performance. Method 2 attained better results than all the other classifiers in all the evaluation metrics (except training time). It misclassified only 2 benign emails as phishing (FPs) and 59 phishing emails as benign (FNs), hence it accomplished a high F1- score (0.9942), 0.9943 accuracy, precision, and recall, as well as 0.9714 AUC and 0.967 MCC. Moreover, it had a very low training time (0.0313 sec) for an Ensemble Learning classification, which was the second-lowest training time after KNN (0.01 sec). Following, Method 1 accomplished 0.9907 F1-score, accuracy, and recall, 0.906 precision, 0.969 AUC, and 0.9466 MCC which was the outcome of 39 FPs and 60 FNs.

Regarding the remaining classifiers, RF outperformed MLP, DT, and KNN. To be more precise, RF attained 0.9856 F1-score, 0.986 accuracy, 0.9805 precision, 0.9808 recall, 0.9323 AUC, and 0.918 MCC, which were the results of 10 FPs and 139 FNs. Thus, RF performed very well on the classification of benign emails and not well on the classification of phishing emails. MLP achieved 17 FPs and 143 FNs, which resulted in 0.9845 F1-score, 0.985 accuracy and recall, 0.9849 precision, 0.93 AUC, and 0.9117 MCC. DT had fewer FNs (120) than RF and MLP, but more FPs (85), hence it accomplished 0.9806 F1-score, 0.9808 accuracy and recall, 0.9805 precision, 0.9376 AUC, and 0.8887 MCC. KNN misclassified 102 FPs and 480 FNs, namely it accomplished 0.9398 F1-score, 0.9454 accuracy and recall, 0.9416 precision, 0.7628 AUC, and 0.6273 MCC. Finally, the ROC curves of all the classifiers are shown in Figure 3-6. Please note that Ensemble Learning as well as the hybrid features are not interpretable, hence the experimental results cannot be thoroughly analyzed and it is not possible to provide justifications about the reasons that led to the misclassifications. However, in the next section, the evaluation results were further discussed considering the ML algorithms' traits and provide answer to the RQ2 (*Can the combination of Ensemble Learning with hybrid features, overall, enhance the phishing email detection performance?*).

Table 11 Classification Results using hybrid features

Classifier	**F1-score**	**Accuracy**	**Precision**	**Recall**	**AUC**	**MCC**	**Training time**	**TP**	**TN**	**FP**	**FN**
LR	0.858	0.9028	0.8609	0.9028	0.503	0.0468	72.998	9611	7	1028	8
GNB	0.8561	0.9014	0.815	0.9014	0.4992	0.0123	0.095	9604	0	1035	15
KNN	0.9398	0.9454	0.9416	0.9454	0.7628	0.6273	**0.01**	9517	555	480	102
DT	0.9806	0.9808	0.9805	0.9808	0.9376	0.8887	0.407	9534	915	120	85
MLP	0.9845	0.985	0.9849	0.985	0.93	0.9117	17.005	9602	892	143	17
RF	0.9856	0.986	0.9805	0.9808	0.9323	0.918	4.038	9609	896	139	10
Method 1	0.9907	0.9907	0.9906	0.9907	0.969	0.9466	13.705	9580	975	60	39
Method 2	**0.9942**	**0.9943**	**0.9943**	**0.9943**	**0.9714**	**0.967**	0.0313	**9617**	**976**	**59**	**2**

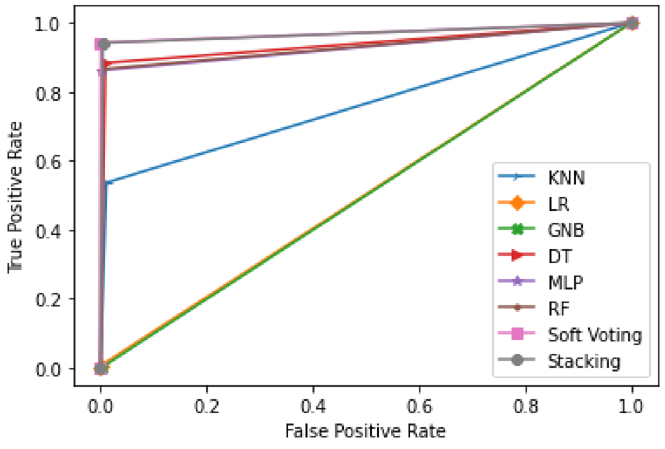


Figure 3-6 ROC Curves

3.4.6.1 Discussion

Based on the tabulated results of Tables 9, 10 and11 the RQ2 can be answered. As one can notice by comparing the results on Tables 9 and 10 with Table 11, the only algorithm that obtained higher detection performance with the hybrid features is MLP. DT achieved slightly worst performance, while the remaining algorithms (i.e., LR, GNB, KNN) attained worst performance. That is because MLP can handle data with very large dimensions (i.e., with a higher number of features) than the other algorithms, hence the hybrid features contribute to the enhancement of its performance. On the other hand, DT mostly focuses on the more informative features to perform the classification, which are the content-based features HTML code, number of different HREF, and number of dots, thus the rest of the content-based features, as well as the text-based features, did not influence its performance. LR, GNB, and KNN were not able to effectively process the high number of hybrid features, since they attained better results when processing the content-based or text-based features separately. More specifically, LR and GNB are not able to effectively process data with large dimensions (i.e., the text-based and hybrid features), while KNN could not effectively process the content-based features (see Table 9) affecting also on its performance when employing the hybrid features. Moreover, the rationale for employing the training time as a metric was the performance issues that may arise due to the combination of hybrid features with Ensemble Learning methods. Thus, it was measured the performance of the proposed methodology using training time and as it is shown on Table 11, which in case of Method 2 is very low in comparison with the other ML/DL models (only the training time of KNN is lower) as well as the training time is similar to ML/DL models when they parse only the content-based (Table 9) or the text-based (Table 10) features. Therefore, it was found that the hybrid features, overall, enhance the prediction results without significantly affecting the performance. Based on the above mentioned, regarding RQ it is evident that the combination of hybrid features with Ensemble Learning, overall, improves the phishing email detection performance (see Table 11), since both Method 1 and Method 2 outweighed all the other algorithms. Thus, it was concluded that Ensemble Learning when combined with hybrid features, overall, enhances the phishing email detection performance.

3.4.7 Comparison of the Proposed Methodology with Existing Works

This section elaborates on the comparison of the results of the proposed methodology with the results of prominent existing works from the period 2011-2022. Please consider that the proposed methodology cannot be compared directly with existing works due to the diversity of the evaluation approach that each work implements (i.e., different samples, class ratios, and metrics). Table 12 depicts the results of the proposed and existing works in terms of the most common evaluation metrics (i.e., accuracy and F1-score), the category of the deployed features (i.e., content-based, text-based, and hybrid), as well as on the number of the utilized benign and phishing samples. Some existing works, such as [71] and [72] revealed very good detection performance; however, their results are not representative of how these works will carry out in real-life implementations since their assessment was performed on a significantly smaller dataset (4,150 benign and 2,279 phishing emails) that is exclusively comprised of obsolete phishing emails. Other works that accomplished good detection performance, like [74] and [78] apart from obsolete phishing emails, they employed inappropriate evaluation metrics to measure the prediction performance (i.e., the accuracy metric that is inappropriate for imbalanced data). Considering that the proposed work was evaluated using a large imbalanced dataset (32,051 benign and 3,460 phishing emails) that considers the evolution of phishing emails and employs a realistic scenario where the phishing emails are much less than the benign, its numerical results with Soft-Voting (i.e., Method 2) are promising.

Moreover, the approach presented in [142] also attained comparable performance on a large dataset. The authors of [142] followed a similar approach with the proposed work to tackle phishing attacks but focused on the detection of phishing websites instead of phishing emails. More specifically, the similarities of [142] and the proposed methodology are that both works deployed ML methods to perform the detection, employed hybrid features, and carried out a thorough evaluation using one large dataset instead of multiple smaller ones. Although there are a few similarities between the proposed methodology and [142], there are more differences (except from the fact that they deal with different phishing domains). The major differences are that (a) the proposed work is based on Ensemble Learning to separately process the hybrid features and obtain higher detection performance and not on a boosting method as in [142] (e.g., XGBoost); (b) the hybrid features of the proposed work are a combination of text-based and content-based features from emails, while the hybrid features of [142] are the merge of URL and HTML features from websites; (c) the proposed work deployed Word2Vec to extract the textual information from emails instead of TF-IDF ([142] uses TF-IDF to extract character level information).

To the best of our knowledge, the proposed methodology attained the highest scores that have been achieved in the phishing email detection domain using such a diverse dataset that considers the evolution of phishing emails. Furthermore, none of the existing works consider the guidelines proposed in Section 3.4.1 (i.e., G1-G5), hence their results might be misleading or biased [15]. Finally, a common drawback of existing works is that they did not share the appropriate information to reproduce their approach. To tackle this drawback, the utilized datasets and the code of the proposed methodology are shared to the public to drive more research in the phishing email detection domain and to facilitate researchers to re-produce the results of the proposed methodology and compare it with future solutions [143].

Table 12 Comparison of HELPHED with Existing Works

Paper/Year	**F1-score (%)**	**Accuracy (%)**	**Feature Category**	**# Benign Samples**	**# Phishing Samples**
Hamid et al. (2011) [59]	-	92	Content	2364	2230
Moradpoor et al. (2017) [61]	-	92.2	Content	6,656	7,714
Akinyelu et al. (2014) [63]	97.91	98.96	Content	1800	200
Islam et al. (2013) [144]	-	97	Content	N/A	N/A
Smadi et al. (2015) [64]	98.09	98.11	Content	4,559	4,559
Alhogail et al. (2021) [69]	98.5	98.2	Text	4,894	3,685
Gualberto et al. (2020) [71]	99.9	99.9	Text	4,150	2,279
Gualberto et al. (2020) [72]	100	100	Text	4,150	2,279
Fang et al. (2019) [74]	99.33	99.84	Text	7,781	999
Hiransha and Nidhin (2018) [76]	-	96.8	Text	5,088	612
Egozi et al. (2018) [78]	99	-	Text	7,689	1,210
Halgaš et al. (2019) [79]	98.63	98.91	Text	6,951	4,572
Unnithan et al. (2018) [82]	98	97	Text	7,781	997
Unnithan et al. (2018) [83]**]**	-	88.4	Text	8,913	1,087
Yadav et al. (2017) [145]	-	98.02	Content	2,550	500
Aljofey et al. (2022) [142]	96.38	98.28	Hybrid	32,972 (web-pages)	27,280 (web-pages)
Proposed (Soft-voting)	99.41	99.42	Hybrid	32,051	3,460

4. A Holistic Methodology for Detecting Exploit Kits

4.1 Introduction

It is widely known that the browser is the main gateway to access the Internet, it continues to evolve to suit the needs of the worldwide web and modern web applications. At the same time browser's functionality is ever-expanding through the use of plugins introducing new computing paradigms. For example, the Chrome OS is an operating system solely based on the Chrome browser. As the complexity and the functionality of browsers increase, it is evident that the attack surface also increases. As a matter of fact, multiple vulnerabilities have been discovered in browsers and significant attacks against insecure plugins have made headlines over the years [146]. Browser vulnerabilities may expose users' data and lead to catastrophic consequences from stealing passwords and credit cards to silently load malware and gain remote access [147]. Thus, it is expected that browsers will continue to be the prime target of cyber-criminals utilizing EKs to exploit vulnerabilities and attack end-users.

Moreover, a recent, large-scale study of EKs functionalities presented in [18] concluded that a common trait of EKs is their ability to quickly adapt to new malware trends. Indeed, nowadays EKs have evolved and shifted their modus operandi to more lucrative methods by delivering ransomware. For instance, the Maze ransomware was initially delivered through spam email campaigns, but cybercriminals changed their tactics by taking advantage of EKs to infect computers. Due to the rise of the popularity of cryptocurrencies, EK authors have also shifted their focus to distribute cryptocurrency mining malware, which is classified as a new attack type called Drive-By-Mining (DBM) to deliver cryptojacking malware [19]. Another interesting finding in [18] was that EKs pose a significant threat, due to their intelligence to bypass contemporary security countermeasures. One such example is the fact that EKs have evolved beyond file-based methods, with malicious files now existing as memory processes in the operating system in order to evade detection (file-less payloads). This makes even more challenging the detection of EKs at the host level as the delivery payload does not touch the disk and leaves no room for antivirus technology to detect EKs [20]. Thus, we conclude that despite their longevity, EKs are still relevant in the underground malware market as they evolve by adapting to new malware trends. A recent report from MalwareBytes [148] published in 2020 highlighted that incidents of EKs are on the rise by a new generation of EKs that have adopted more sophisticated techniques to evade security mechanisms and distribute malware in covert ways. According to McAfee, many new EKs such as Spelevo, Capesand, Lord, Fiesta, and Fallout were discovered in 2019. At the time of writing this work, a new EK has emerged named Purple Fox that includes several 1-day exploits such as CVE-2021-26411 and CVE-2020-0674 [51]. Overall, this thesis argues that the problem of EKs still poses a viable threat to both individuals and organizations.

Moreover, EKs that deploy file-less malware are harder to detect, since they do not leave detection traces at the file system and resides only in the host memory. Besides, other evasion techniques can be applied including obfuscation and encryption to bypass host based defensive mechanisms. Thus, new effective detection technologies are needed to curb the ongoing threat of EKs. Considering the above observations, this thesis will seek answers to the following research question:

Can we build accurate EK detection systems that are capable of efficiently detecting state-of-the-art EKs based on traces at the network level operation of EKs? (RQ3)

To answer this question this thesis makes the following contribution:

A holistic ML-driven methodology for the detection of Exploit Kits exploiting the network traces they leave behind during an infection.

More specifically, during this thesis a Machine Learning (ML) methodology, named EKnad (Exploit Kits' network activity detection) has been conceptualized and designed that can detect EK infection activity by analyzing the network traffic from packet captures. To be more precise, EKnad contains two phases, the *Pre-processing* and the *EK detection*. *Pre-processing* separates the HTTP communication from the rest of the network traffic and performs the creation of potential EK sessions procedure. In the creation of potential EK sessions, the HTTP flows are grouped in order later to identify faster and more accurate the EK redirection chains. Next, the *EK detection* phase performs the feature extraction, where discriminating features are extracted from the potential EK sessions. More specifically, a comprehensive set of 47 features was proposed that include all the infection procedure of EKs and is divided into 5 categories: Header, Redirection, Evasion, URL & Domain, and Connection. Emphasis was put on the EKnad evaluation, where the network traffic from 26 different EK families was deployed to conduct four realistic evaluation experiments. The evaluation methodology was created carefully to comply with state-of-the-art guidelines that should be considered in the evaluation of ML classifiers to avoid misleading and erroneous results. The evaluation results indicate that the Multilayer Perceptron (MLP) Artificial Neural Network classifier is the best selection for EKnad, since it accomplished the best detection performance outperforming other well-known ML algorithms (i.e., Random Forest, Bayesian Network, Decision Table, k-Nearest Neighbors, J48). Finally, based on the thorough assessment this thesis concluded that the proposed methodology is able to: a) detect new and last-generation EKs based on the training of older EKs, b) detect less-known EK families based on the training of well-known EK families and c) outperform the famous rule-based IDS Snort and Suricata.

The rest of this Chapter is structured as follows. Section 4.2 provides background information on EKs along with their impact in the cybercrime scene, while section 4.3 thoroughly discusses the existing works on the detection of EKs. Section 4.4 elaborates on the network activity of EKs and introduces network-level features. Section 4.5 presents the proposed methodology, where sections 4.5.1 and 4.5.2 discuss the pre-processing and the EK detection stages respectively. Section 4.6 analyzes the experimental evaluation of EKnad focusing on the various realistic evaluation scenarios. Please note that parts of this Chapter have been published in [32].

4.2 Background on Exploit Kits

Exploit Kit (EK) is typically a malicious toolkit used to exploit system vulnerabilities and spread malware. This study exclusively considers EKs that are designed to exploit web browser and plugin vulnerabilities automatically and invisibly. EKs are monetary-centric frameworks built to execute drive-by-download (DBD) attacks [51], which can contaminate computerized systems with any form of malware (e.g., keyloggers, trojans, etc.) when browsing a website. The victim is not required to take any action (such as clicking a link) beyond visiting the website; the DBD attack will be carried out in the background. The EK creators (usually refereed as authors) use a variety of methods to entice victims to visit the EK's website. The EK authors utilize several methods to lure a victim to visit the EK's web page [21]. In particular, the three major infection vectors are:

1. Malvertising: On prominent websites, harmful adverts are referred to as malvertising. The EK creators can create accounts that let users to upload advertisements with bespoke designs, which are then displayed online by the advertising provider. They meticulously embed malicious code within the adverts so that visitors are secretly redirected to the EK website.
2. Compromised web pages: EK authors scour the internet in search of websites with inadequate protection in order to exploit them and inject malicious code into their web server. The malicious material may be, for example, an HTML element known as iframe (inline frame) that redirects the unknowing visitor to the EK website.
3. Social engineering: EK authors deploy social engineering techniques, like phishing or spear-phishing emails that contain an attachment or a link that points to the EK web page.

EKs are software applications that provide computer systems with an automatic exploitation environment. Without attracting the victim's notice, they identify and exploit web browser vulnerabilities to deliver dangerous payloads (i.e., malware) [22]. By building a business model akin to the software-as-a-service paradigm, the designers of EK have advanced the DBD attack process. Specifically, EK clients can acquire subscriptions on a pay-per-install basis and select from a variety of features (e.g., number of exploits or malware) [23] to harm personal computers via the browser. The prevalence of EKs is heavily influenced by their price, distribution network, and functionalities. Additionally, EKs emphasize user-friendliness to attract more customers. For example, they offer a management console with a graphical user interface that automates the exploitation and infection processes; hence, EK users do not need to be computer experts.

In the following, the EK workflow is described (see Figure 4-1): (1) The procedure starts with the attacker that compromises a legitimate web page by placing malicious content (or sets up a malicious website); (2) The victim visits the compromised web page and the browser renders the malicious content; (3) The malicious content silently redirects the victim to the EK server. The EK server fingerprints (i.e., gather information about the operating system, browser version, plugins, geolocation, and language) the victim browser to determine if it is vulnerable and selects the appropriate exploit; (4) Based on the identified vulnerabilities, a suitable exploit is selected and is delivered to the victim's browser (if there are multiple vulnerabilities, multiple exploits may be executed); (5) In case of successful exploitation, the victim's browser replies accordingly. In case of unsuccessful exploitation, the EK server delivers the next suitable exploit; (6) The browser unwillingly downloads what is known as a "payload", which is the malware binary file; (7) Lastly, the malware is executed, and the host is infected.

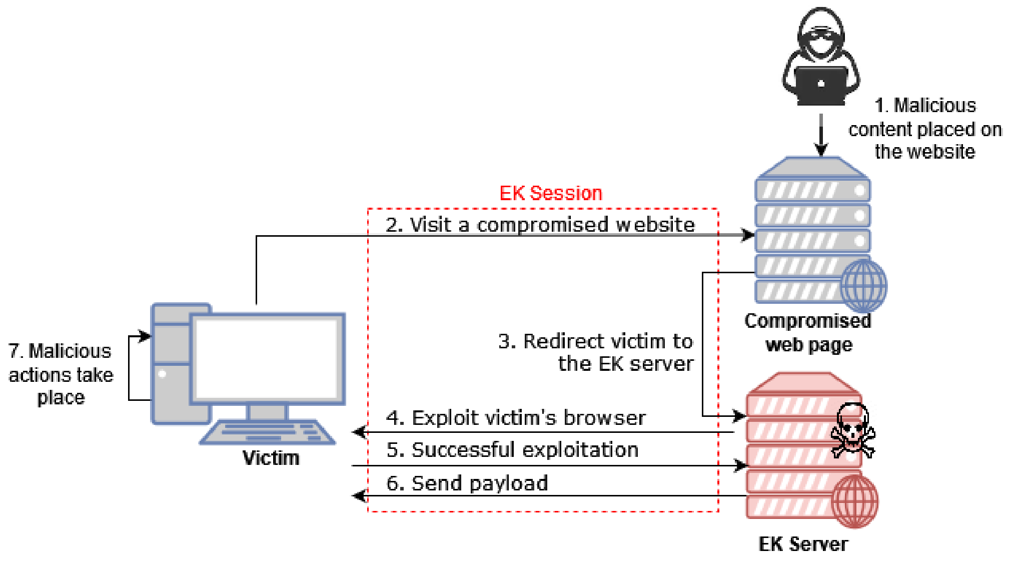

1. Malicious content placed on the website
EK Session
2. Visit a compromised website
Compromised web page
3. Redirect victim to the EK server
7. Malicious actions take place
Victim
4. Exploit victim's browser
5. Successful exploitation
6. Send payload
EK Server

Figure 4-1: Exploit Kit Infection Procedure

Moreover, EKs employ several advanced features to evade detection mechanisms, making them stealthier and hiding their infection activity. These features are:

1. HTTP 302 cushioning: Some EKs (e.g., Blackhole, Angler, Neutrino) utilized a technique known as HTTP 302 cushioning where the EK authors exploit the legitimate HTTP 302 response to create a sequence of redirects through proxy domains before the victim is redirected to the EK landing page (the first page that the victim visits in the EK server). This technique removes the need for iframes, which are heavily used by EKs to redirect the victim to the EK landing page, and relies on a legitimate browser feature hardening the EK detection process.

2. Obfuscation: EKs employ obfuscation techniques to their code to avoid detection by malware scanners. The obfuscation techniques that EK deploy are randomization, string obfuscation, string concatenation, character substitution, polymorphic obfuscation, etc. [52]. The obfuscated code is updated frequently, as the EK authors check whether the signature of the obfuscation scheme is already known. An interesting fact is that some EKs authors use commercial obfuscation software (e.g., IonCube) to protect the EK source code [53].
3. Cloaking: Many EKs advanced the fingerprinting process by inspecting the IP address and the system information to target only certain countries or identify addresses that are related to honeypots and security vendors. In case a victim does not meet the selected criteria, the EK process will not take place and it will redirect the victim to a legitimate web page or it will display an HTTP 404 error. In addition, if the system is not vulnerable the EK will redirect the victim to a legitimate web page [54].
4. Blacklist avoidance: EKs deploy Domain Generation Algorithms (DGA) to change the domain names constantly. This is done periodically or when an IP/URL is disclosed to a known blacklist. Recent EK authors deployed a technique named “Domain Shadowing”, where they hijack domain registrant accounts and create many seemingly random subdomains [55].

4.2.1 Exploit Kits Impact on Cybercrime scene

Multiple EKs are regarded as successful since they have proven beneficial for hackers. Blackhole was the first advanced EK to embrace the rental business model and a benchmark for subsequent EKs. Its defining characteristics were advanced evasion tactics (server-side polymorphism, JavaScript & HTML obfuscation, and string manipulation techniques). In addition to Blackhole, other notorious EKs evolved, such as Nuclear and Angler. Rig EK has been one of the most successful EKs since its introduction in 2014 and is still operational at the time of writing. In 2015, Rig was responsible for infecting over 27,000 computers every day[28]. Its low price upon initial release contributed to its rapid rise in popularity. Rig utilizes the business model of Blackhole, and its monthly subscription costs range from $700.00 to $2,000.00. The Rig EK utilized domain shadowing, whereas Rig mostly delivers malware and cryptocurrency miners today. New EKs continue to develop [56] , indicating that EK activity is tough to eradicate. The Fallout EK is a very recent EK, having debuted in August of 2018. It is served through malvertising campaigns that mostly affected users in Japan, Korea, the Middle East, and Southern Europe. Fallout EK targets Internet Explorer and Flash Player vulnerabilities (e.g., CVE-2018-8174 and CVE-2018-15982), while its malware list consists of ransomwares, banking trojans, and information stealers. Spelevo EK was first discovered in mid-2019 and delivers different banking trojans (e.g., IcedID and Dridex). Lord EK is also one of the most recent EKs, as it was discovered in August 2019, and it distributes the ERIS ransomware and the njRAT trojan. Purple Fox is the name of the most recent EK, which surfaced in 2021, and it includes various browser flaws such as CVE-2021-26411 and CVE-2020-0611.

4.3 Related Work

The existing works that employ ML/DL methods to detect EKs can be classified in three categories based on the data source (e.g., network traffic) that utilize to detect the EKs: a) Network Traffic (i.e., logs, PCAPs), b) EK server-side source code (i.e., PHP), c) EK client-side source code (i.e., HTML, JavaScript, etc.).

4.3.1 Network Traffic

In the network traffic category, Mekky et al. [88] were among the first that exploit the HTTP redirection chains for the detection of DBD attacks (upon which EKs are based). To achieve this, the

[28] https://www.trustwave.com/en-us/resources/blogs/trustwave-blog/how-an-upgraded-version-of-the-rig-exploit-kit-is-infecting-27k-computers-per-day/

authors built a per-user browsing activity tree, which maintains all the redirection chains that are triggered by an HTTP request. Moreover, they deploy a Decision Tree classifier (J48) to detect the malicious HTTP redirection chains. The experiments showed high detection accuracy from 99.1% up to 99.9%. An extension of the above work presented in [89], in which the HTTP packets are dissected and reassembled into bidirectional TCP flows to perform tree-based similarity searches. The authors concluded that their method outperforms the state-of-the-art in terms of false positive and false negative rates. Both of the aforementioned works rely solely on the redirection URLs to perform the detection. Moreover, the dataset in [88] is limited while in [89] the authors do not take advantage of ML and overlook to consider several other characteristics of EKs.

Towards this direction, Nikolaev et al. [90] present a method that detects EKs' by deploying traits entirely from HTTP proxy logs. Five traits were extracted from the HTTP proxy logs and fed in the detection algorithm and used regular expressions to classify the malicious activities in one of the following EK families: Angler, Neutrino, and Rig. The results were promising, however, the authors relied their evaluation only on three EK families. Moreover, the extracted traits did not take into account the whole infection procedure of EKs, as well as they did not consider an important trait of EKs, namely the chain of redirections, as they concentrate on a single malicious HTTP request/response pair to detect EKs. Qin et al. [53] relied on HTTP header information to built a graph model from which they extracted EK detection features including several graph-specific features such as node centrality, longest path, etc. The evaluation performed on a dataset that contains 9362 EK and 10541 benign network traffic files using 10-fold cross-validation. The Random Forest classifier accomplished the best detection performance, namely 94.9% accuracy. The deployed dataset contains a large number of EK PCAP files (9,362 in total) from two websites, (1) malware-traffic-analysis.net (993) and (2) threatglass.com (8,369). Although the proposed solution attains high detection accuracy, the utilized dataset in the experiments has several pitfalls. In particular, several EK PCAP files from malware-traffic-analysis.net website [91] include the same EK running in a different environment, which means that the obtained dataset does not include unique samples. Furthermore, the EK PCAP files of the threatglass.com website (which is not accessible anymore, hence the 8,369 EK samples cannot be verified) may appeared also in [91], namely duplicated samples may be included in the dataset. Also, the authors did not provide any information regarding the utilized EKs families neither whether they tried to detect EKs from the same family, fact that may significantly improve the detection results, since EKs of the same family do not have many differences. Finally, the ratio between benign and EK samples of the training and testing datasets is approximately the same. This balanced dataset does not represent a realistic scenario since benign network traffic is higher than malware traffic with EKs.

In [92] the authors proposed an ML approach for the classification of EKs based on EK network traffic files (PCAP). The EK files were used as input into the Zeek (known as Bro) IDS to create network flows and extract from them various content-based, interaction-specific, and connection-specific features from the HTTP, DNS, and File logs. They deployed a Decision Tree classifier, which detected the EK traffic and the EK families with 97.74% accuracy. The paper does not evaluate other classifiers, while the number of the considered EK families is limited to five. Towards this direction, Burgess et al. [93] also deployed the Zeek IDS to extract HTTP redirection chains from EKs network traffic. In their study, the authors identified nine redirection techniques and they were able to extract 96.52% of the EK domains. The authors' main goal was to generate HTTP redirection features and not to propose a full-fledged EK detection solution. In their newer work, Burgess et al. [52] deployed the 1,279 EK and 5,910 benign redirection chains that were identified in [92] to detect EKs using an LSTM RNN classifier. The classifier attained 0.9878 F1-Score using only redirection features. The authors did not provide a comprehensive solution since they focused only on one trait of EKs (i.e., the redirection) as well as they did not compare the proposed classifier with other well-known ML classifiers to validate its detection performance. Furthermore, it is unclear from how many EK PCAP files the authors concluded to 1,279 EK redirection chains.

4.3.2 EK Server-side Source Code

The second category is comprised by fewer research works that focus on the EKs' server-side source code to extract information that can be deployed by EK detection systems. Eshete et al. [94] proposed WebWinnow, a system that deploys supervised ML models to detect whether a given URL is hosting an EK. WebWinnow deploys in total 30 features extracted from the source code of 38 EKs. These features were separated into 4 categories: Aggregate-Behavioral, Interaction-Specific, Connection-Specific, and Content-Specific. The authors claim that WebWinnow using a Decision Tree (J48) classifier achieved 99.9% true positives and 0.1% false positives in the testing phase. On the other hand, in PExy [95], the authors applied static analysis on the EKs' source code to examine how the utilized exploits will be triggered. PExy was able to provide all the necessary conditions to provoke all exploits and detect all the paths to malicious output from EKs with very few false positives and false negatives. Finally, EkHunter [94] is a toolkit that aims to compromise EK servers by performing a vulnerability analysis on the source code of EKs. The authors surveyed 30 EKs and found over 180 vulnerabilities in 16 EKs, while they managed to exploit 6 of them. A fundamental limitation of the above-mentioned research works is the scarce availability of EKs source code, as the only way to obtain it is when the authorities cease the EK activities and release the EK source code.

4.3.3 EK Client-side Source Code

In the third category, are described existing works that examine the client-side source code dealing mostly with the malicious JavaScript code that is heavily deployed by DBD attacks and EKs. Curtsinger et al. [96] proposed Zozzle, a JavaScript malware detector that is deployed in the browser. Zozzle combines a JavaScript Abstract Syntax Tree (AST) with a Naive Bayes classifier. The experimental results showed 99.45% detection accuracy and a very low FPR. CUJO [97], is a detection and prevention system for DBD attacks that transparently inspects web pages and blocks the delivery of malicious JavaScript code. CUJO was implemented as an extension to a web proxy and combines static and dynamic analysis of the JavaScript code and a Support Vector Machines (SVM) classifier to detect malicious code. Both Zozzle and CUJO focus only on the JavaScript code to extract detection features which is a challenging task since EK authors constantly try to discover new obfuscation techniques to hide their code. Based on the same principles, Canali et al. [98] presented Prophiler, a lightweight filter for DBD attacks, which deploys static analysis techniques to quickly examine a web page for malicious content. Prophiler filters non-malicious pages so that malicious pages can be examined more extensively by honeypots. To evaluate Prophiler, the authors used the Random Forest classifier for HTML features, whilst for JavaScript & URL and host-based features, they deployed a Decision Tree classifier (J48). The proposed solution can be considered as a filter that performs an initial check on web pages rather than a detection system.

FriSM [99] is a detection model for EKs based on string-similarity matching. The authors introduced features belonging to three classes: probabilistic (e.g., number of words, JavaScript functions, etc.), size-based (e.g., web page file size, string size, etc.), and distance-based (e.g., the ratio of alphanumeric characters, code string similarity, etc.). The aforementioned features were extracted from the EKs network traffic and deployed to measure whether they are similar to patterns generated by EKs activities. Although the results were promising, FriSM is sensitive in terms of false positives and the string-similarity matching technique requires a similar structure between different EK families to be accurate. In case that the EKs' structure changes, the extracted features will not be informative, and patterns cannot be created. In [21], the authors introduced a visualization approach for the detection of EKs. In particular, the EK source code (the paper claims that both client-side code such as JavaScript and server-side code such as PHP are utilized) were converted into gray-scale images and the authors observed that different EK families have different code schemes, which result in different images. A convolutional NN classifier was used to detect the EKs. With this approach, the authors accomplished very high detection performance combined with low detection time. However, a general limitation of malware detection based on image classification is that it can be easily evaded by adding garbage code.

4.3.4 Limitations of existing works

To summarize, the works of the network traffic category utilize a limited number of features to detect EKs. In the server-side source code category, a fundamental limitation is that the source code of EKs is not publicly available, thus it is difficult for these works to be applied in a generic manner. In the client-side source code category, the authors focus on the JavaScript and HTML code for EKs detection, which is easily evaded. However, EKs can easily change their obfuscation techniques making detection on the source code level very hard. Overall, when it comes to the evaluation, previous works (regardless of their category) share common limitations, such as considering a small number of unique EK families in the datasets or experiments performed on balanced datasets that are not representative of EKs in the wild.

To address the identified limitations of the existing works, in this thesis it has been developed a holistic methodology, named EKnad, for the detection of EKs employing a thorough feature set that is extracted from PCAP files. By experimenting with numerous ML algorithms using representative samples of various EK families (i.e., 26 different EK families), EKnad managed to both detect old and recent EKs using imbalanced datasets in multiple evaluation scenarios.

4.4 Exploit Kits Network Analysis and Proposed Features

To detect EK activity promptly and accurately, this thesis argues that the network traffic includes a wealth of information that can be utilized for the detection of EK presence. Particularly, the intuition of this thesis is that EKs follow an attack pattern, that is a victim is redirected in a compromised website, and subsequently redirected to an EK server, which initiates the exploitation of the victim's browser and concludes with the payload delivery. This sequence of events leaves behind tracks on the HTTP protocol from which discriminating features of EKs activities can be extracted. Based on the above observations the following 5 feature categories are defined:

1. EKs generate HTTP requests and responses, which employ specific header features to perform specific malicious actions. For instance, to exploit a Java vulnerability in the browser, they deploy the Content-Type: application/ java-archive header.
2. Before delivering the malware, EKs perform multiple redirects to infect a victim's computer. Consequently, several redirection features are generated. Previous works have also considered this EK function [149], [88].
3. Elaborating more on the HTTP traffic, we observed that EKs hide their presence by deploying evasion features, which mainly refer to the encoding of the web pages' contents.
4. EKs' web pages contain URL & Domain features that are equivalent in all EK families and different from the benign web pages (e.g., the EKs' domain names entropy appeared to be higher than the benign domain names).
5. Finally, connection features are a part of EKs' network activity. We noticed that as EKs perform specific actions, hence they generate a specific number of HTTP GET and POST requests. In contrast, in benign activity this number is arbitrary. Consequently, the average number of HTTP requests and responses differ in EK and benign network activity.

The aforementioned 5 feature categories are extracted from all steps of EK infection procedure (i.e., Steps 2-6 of Figure 4-1). However, it is important to mention that there are some specific features that are based on the EK redirection procedure (i.e., Step 2 of Figure 4-1) such as Location Header or Referer header, while other features are exclusively based on the EK exploitation procedure (i.e., Steps 4-6 of Figure 4-1) such as Content-Type: application/java-archive. For clarity, the features that are utilized only in the exploitation steps are highlighted in the description below. It should be also mentioned that some of the features have been also considered in the literature [92], [93], [150].

The proposed EK detection features are analyzed based on the above 5 feature categories. The Header Features category is comprised of HTTP headers that appeared in the HTTP requests and responses between the EK and the victim. The Redirection Features category encompass various features, such as HTTP redirection, URL parameter redirection, etc., which are occurred during the redirect from the compromised website to the EK server. The Evasion Features category represents the advanced obfuscation techniques that EKs employ. The URL & Domain Features are based on suspicious traits that appear in the format of the URLs and in the domain names as utilized by EKs [151]. In the HTTP traffic, there observed patterns also in URLs as well as domains associated with EKs. The Connection Features category is comprised of features of the HTTP traffic in general (e.g., number of HTTP GET requests, number of HTTP responses, etc.). Please note that the proposed features can be also present in benign traffic. For example, the Content-Type: application/octet-stream feature (see below) is present in both benign and EK traffic. For this reason, none of the proposed features can be used standalone for detection. Moreover, some of the proposed features are (or have been) utilized frequently by several EKs. For example, the Content-Type: application/java-applet has been utilized for the exploitation of Java applets by EK families, such as Blackhole, Neutrino, and Nuclear. However, EKs do not rely only on one specific exploit and continuously try to adapt their exploitation strategy according to the latest exploits available. This also corroborates the combination of multiple features for detection. Below the proposed features for each feature category are analyzed. Please note that the notation used to count the features includes the first letter of the feature category followed by two letters of the subcategory (if there is any subcategory) followed by the digit counter. For example, the H.CT.1 is the first feature of the Header (H) category that includes the Content Type (CT) subcategory.

Header Features: This feature category contains only binary features, namely whether the features are presented in the network traffic.

H.CT Content-Type:

H.CT.1 text/html: The text/html header is used to inform the browser that the content of the served web page is HTML.

H.CT.2 application/java-archive: This header refers to Java archive (jar) files and is usually deployed by EKs to send a Java exploit in the browser. Hence it is utilized in the EK infection step 4 as depicted in Figure 1.

H.CT.3 application/javascript: The application/javascript header is used to understand the browser that the received content contains JavaScript code.

H.CT.4 application/java-applet: This feature indicates that the HTTP message delivers a Java applet, which is located on the EK server and is executed on the web browser of the victim. Some well-known EK families, such as Blackhole, Neutrino and Nuclear, utilized Java applets to exploit Java vulnerabilities (e.g., CVE-2012-468 and CVE-2013-2463). The Content-Type: application/java-applet header is utilized in the EK infection step 4 as presented in Figure 4-1.

H.CT.5 application/octet-stream: This HTTP header is used to indicate that an HTTP message contains arbitrary binary data. This header is often used by benign websites to convey data to the browser such as applications, documents, images, etc. When it comes to EKs, they have utilized it to deliver malware in the form of a Windows executable files (.exe). Typically, EKs combine the headers Content-Type: application/octet-stream and Content-Disposition: attachment (H.CD), in order to directly download malware into the victim’s computer. The Content-Type: application/octet-stream has been applied also in previous works for the detection of EKs [150]. This feature represents the EK infection step 6 as highlighted in Figure 4-1.

H.CT.6 application/xml: This Content-Type header is used to serve xml data in the browser. It was observed that this header is used by various EK web pages.

H.CT.7 application/x-shockwave-flash: This header indicates that the web page transmits a Small Web Format (swf) file, which is basically an Adobe Flash file. It was noticed that many EKs target Adobe Flash vulnerabilities and send swf files to exploit them. This header is used in the EK infection step 4 as shown in Figure 4-1.

H.CT.8 application/x-silverlight-app: This header is Microsoft's equivalent of the application/x-shockwave-flash header and it is used to send .xap files.

H.CT.9 application/postscript: The application/postscript header is used to deliver a PostScript (ps) file in the browser. EKs deploy ps files to exploit the related victim's browser plugins.

H.CT.10 application/pdf: This header is utilized to serve malicious pdf files to exploit related browser plugins.

H.CT.11 application/x-msdownload: This header is commonly used to indicate whether a DLL file is presented in an HTTP response message and it delivers .exe files. It was observed that some EK families deploy the x-msdownload header to deliver a malware, hence this header is employed in the EK infection step 6 of Figure 4-1.

H.CT.12 application/x-msdos-program: This header is deployed to denote the presence of an "MS DOS" application or executable file in an HTTP response. EKs usually employ this header for malware delivery purposes.

H.CT.13 application/x-dosexec: This header is a subtype of Content-Type: application/x-msdownload and it is also used to deliver .exe files.

H.UA User-Agent: EKs only target browsers and their plugins [152], hence the HTTP traffic that does not contain a User-Agent header that points to a known web browser (e.g., Internet Explorer, Google Chrome, Firefox, Safari, etc.) it is not related with EK activity.

H.CD Content-Disposition: attachment: When this header is used in an HTTP response, it prompts the user to save the response of a web page as a file. During the analysis of the EKs' network activity, it was noticed that this header is deployed by EKs along with H.CT.5 to download a file in the victim's computer. Thus. this header is also deployed in the EK infection step 6 as illustrated in Figure 4-1.

H.CE Content-Encoding: This header is used to compress the media type of the HTTP message.

***Redirection Features*:** In this category, we extracted features from the URL and the Header fields of the HTTP traffic that are related to redirects from one webserver to another. We did not extract features from the web page contents as they can be obfuscated and reduce the detection performance of the classifier.

R.1 HTTP Redirection: HTTP redirects are commonly used by web developers to redirect the user to a different URL than the one they initially tried to access. Modern browsers automatically and transparently redirect their users when they receive a 301, 302, 303, 304, or 307 HTTP response.

R.2 Location Header: The Location header field contains the URL that the user is going to visit after the redirection. This field forces a web browser to load a different web page than the one that is already opened. The Location header was identified in the network activity of several EKs.

R.3 Referer Header: The Referer (originally a misspelling of the word "referrer") header identifies the address of the web page that originates the request. In case that the domain in the Referer header is not the same with the domain in the HTTP Host header, then the request came from a different domain. The presence of a Referer header was extracted that includes a different domain than the one in the Host header.

R.4 URL Parameter Redirection: The last redirection feature that was extracted from the HTTP traffic is the URL parameter-based redirection. Web pages could redirect a browser to a different web page via a URL parameter. This feature is also deployed in [98].

Evasion Features: A strong trait of EKs is their robustness, which is achieved with evasion techniques. Two binary features were identified that EKs deploy to obfuscate their activity.

E.1 JavaScript Obfuscation: 95% of EKs deploy obfuscation techniques to their JavaScript code [150]. To detect obfuscated JavaScript code, we searched in the EK network activity for the three most common techniques deployed for JavaScript obfuscation. These techniques are:

- Randomization Obfuscation. It replaces the names of variables and functions with randomly created sequences of characters that have no particular meaning, thus we searched for an unreadable sequence of characters.
- Encoding Obfuscation. It converts parts of the code into hexadecimal representations, namely, it encodes the variable's values and the inputs that are passed into functions. Hexadecimal representations are sought.
- Dead Code Insertion. It adds arbitrary code that is executed during the execution of the initial code without affecting its semantics, hence we look for arbitrary code parts.

E.2 HTML Obfuscation: It was observed in the network traffic that EKs utilize also obfuscation techniques to hide the HTML code. The most common techniques for HTML obfuscation are the HTML entities encoding and the base64 encoding. We searched for the aforementioned techniques in HTTP responses that contain the Header Content-Type: text/html.

URL & Domain Features: EKs' network traffic utilizes URL and domain names features that deviate from benign network traffic. This category is comprised of binary, integer (by counting the occurrence of a feature), or a real number value.

U.1 Max URL length: The length of the URL could provide insights regarding whether the URL is benign or malicious. To this end, the maximum length of the URLs that appeared in the network activity was extracted as a feature.

U.2 Max URI directory depth: This feature indicates the maximum directory depth extracted from the URIs of the network activity.

U.3 IP address in the URL: It is common for malicious and EK web pages to be associated with an IP address instead of using a domain name. This feature tracks if instead of a domain name, an IP address is present in at least one URL of the network activity.

U.4 Suspicious filename in the URL: This feature depicts the presence of a randomly generated string or a common filename inside the network activity. Many EKs utilize automatically generated filenames, which are composed of random strings. Moreover, EKs share common filenames. A list was created with these filenames, which later deployed to find associations with the filenames in the URLs of the network activity.

U.5 Port number in the URL: Researchers stated that benign web servers often use the standard port 80, whilst the malicious ones mostly deploy other ports [153]. Thus, we search for different port numbers than the standard ones (80, 443) in the URLs of the network activity.

U.6 Maximum number of occurrences of the "+" character in the URL: The plus symbol indicates that there is a whitespace in the URL. We count the occurrences of this char-acter in the URLs and as a feature it was extracted the maximum number.

U.7 Maximum number of occurrences of the "&" character in the URL: The URLs to deploy multiple parameters utilize the ampersand character. The occurrences of this character were counted in each URL of the network activity, and as a feature it was deployed the maximum number.

U.8 Maximum number of occurrences of the "-" character in the URL: Many URLs contain the dash character, however, the malicious ones, since they are more complicated, usually comprise more dashes than the benign ones. We calculate the number of dashes in each URL of the network activity, and as a feature it was utilized the maximum number.

U.9 Maximum number of occurrences of the "/" character in the URL: The slash character, when used in the URL of an HTTP request, depicts the requested path to the server. Usually, malicious web pages host malicious files in deep paths. We keep track of the number of occurrences of this character in each URL of the network activity, and as a feature, it was deployed the maximum number.

U.10 Jar file: Instead of the application/java-archive header, it is also recorded whether a jar file is presented in a URL. This feature is employed in the EK infection step 4 of Figure 4-1.

U.11 Jnlp file: JNLP (Java Network Launching Protocol) contains information such as the remote address to download a jar file and it is used to launch and manage Java pro- grams on the web. To this end, the presence of a jnlp file is also extracted from the URLs.

U.12 Swf file: The swf (ShockWave) file is an animation created with Adobe Flash and can be played by a browser that has the Flash plugin installed. In addition to the application/x-shochwave-flash header we search in the network activity for URLs that contain swf files. This feature is deployed in the EK infection step 4 as presented in Figure 4-1.

U.13 Exe file: EKs mostly target Windows systems and the majority of them deliver malwares in the form of exe files. Thus, this feature is used in the EK infection step 6 as presented in Figure 4-1.

U.14 Xml file: The xml files have been used by EKs for exploitation purposes, thus it is considered as a feature.

U.15 Mp4a-latm file: EKs utilized latm files, which are associated with audio data, to deliver the malware.

D.1 WWW subdomain in the URL: Malicious web pages usually do not specify a subdomain name [98]. It was observed that various EKs follow this trend.

D.2 Subdomain: Malicious actors usually deploy subdomains to perform malicious actions. Thus, as a feature, it was extracted the presence of subdomains in network activity.

D.3 Suspicious TLD: We created a list of the most known Top-Level Domains (TLDs) for malicious activity, based on [154]. This list is later deployed to identify TLDs related with malicious activity in the network activity and we extracted it as a feature.

D.4 Domain/subdomain entropy: Bigger and more complex domain names appear to have higher entropy than smaller and simpler ones. Entropy is an efficient way to identify domains that have been algorithmically generated (e.g., DGA) and includes arbitrary characters [152] or domains that append dictionary words. EKs usually contain domains with high entropy.

D.5 Domain/subdomain with uncorrelated words: EK's domains use randomly generated domains by appending different words from a dictionary to seem more benign from domains that are algorithmically generated. In this case, the formed domain name consists of words that have no relation to each other. We extracted whether the network activity comprises this domain name form.

D.6 Domain/subdomain length: The domain name length is a feature that has been used before in various researches and has been proved to be effective. Hence, we calculate the domain and/or subdomain name lengths of the network activity and extract the maximum number.

D.7 Suspicious domain name: This feature keeps track of suspicious domain names. As a suspicious domain name, we define: (a) a domain name that is comprised of arbitrary characters that compose an unreadable string, and (b) a domain name that is registered in the list[29].

Connection Features: This category includes features that describe the network activity related to the HTTP requests and responses. In this category, the features are represented by an integer and real number values.

C.1 Number of HTTP GET requests: In this feature, we count the number of HTTP GET requests that appeared in the network activity.

C.2 Number of HTTP POST requests: Similar to the previous feature, we extract also the HTTP POST requests.

C.3 The average length of HTTP response: We calculate the average number of HTTP responses in the network activity and extract it as a feature. All the features that were discussed in this section along with their value type are presented in Table 13.

Table 13 Extracted Features. The bold font indicates the most valuable features based on feature selection

Feature Category	#	Feature name	Type	Description
Header Features	**H.CT.1**	**Content-Type: text/html**	**Binary**	**Presence of text/html header**
	H.CT.2	**Content-Type: application/java-archive**	**Binary**	**Presence of application/java-archive**
	H.CT.3	**Content-Type: application/javascript**	**Binary**	**Presence of application/javascript**
	H.CT.4	Content-Type: application/java-applet	Binary	Presence of application/java-applet
	H.CT.5	**Content-Type: application/octet-stream**	**Binary**	**Presence of application/octet-stream**
	H.CT.6	Content-Type: application/xml	Binary	Presence of application/xml
	H.CT.7	**Content-Type: application/x-shockwave-flash**	**Binary**	**Presence of application/x-shockwave-flash**
	H.CT.8	Content-Type: application/x-silverlight-app	Binary	Presence of application/x-silverlight-app
	H.CT.9	Content-Type: application/postscript	Binary	Presence of application/postscript
	H.CT.10	Content-Type: application/pdf	Binary	Presence of application/pdf
	H.CT.11	**Content-Type: application/x-msdownload**	**Binary**	**Presence of application/x-msdownload**
	H.CT.12	Content-Type: application/x-msdos-program	Binary	Presence of application/x-msdos-program

[29] www.malwaredomainlist.com

Feature Category	#	Feature name	Type	Description
	H.CT.13	Content-Type: application/x-dosexec	Binary	Presence of application/x-dosexec
	H.UA	User-Agent	Binary	Absence of User-Agent header
	H.CD	Content-Disposition: attachment	Binary	Presence of Content-Disposition: attachment header
	H.CE	**Content-Encoding**	**Binary**	**Presence of Content-Encoding header**
Redirection Features	R.1	HTTP Redirection	Binary	Presence of HTTP 3XX code
	R.2	Location Header	Binary	Presence of Location header
	R.3	Referrer Header	Binary	Presence of Referrer header
	R.4	**URL Parameter Redirection**	**Binary**	**Presence of URL parameter that contain an external URL**
Evasion Features	**E.1**	**JavaScript Obfuscation**	**Binary**	**Presence of obfuscated JavaScript code**
	E.2	HTML Obfuscation	Binary	Presence of obfuscated HTML code
URL & Domain Features	U.1	Max URL length	Integer	Maximum URL length
	U.2	Max URI directory depth	Integer	Maximum URI directory depth
	U.3	**IP address in the URL**	**Binary**	**Presence of an IP address**
	U.4	**Suspicious filename in the URL**	**Binary**	**Presence of a suspicious filename**
	U.5	Port number in the URL	Binary	Presence of a port number
	U.6	Max no. of “+” in the URL	Integer	Max no. of occurrences of the “+” in the URLs
	U.7	Max no. of “&” in the URL	Integer	Max no. of occurrences of the “&” in the URLs
	U.8	**Max no. of “-” in the URL**	**Integer**	**Max no. of occurrences of the “-” in the URLs**
	U.9	**Max no. of “/” in the URL**	**Integer**	**Max no. of occurrences of the “/” in the URLs**
	U.10	**Jar file in the URL**	**Binary**	**Presence of a .jar in the URL**
	U.11	Jnlp file in the URL	Binary	Presence of .jnlp in the URL
	U.12	**Swf file in the URL**	**Binary**	**Presence of .swf in the URL**
	U.13	Exe file in the URL	Binary	Presence of .exe in the URL
	U.14	Xml file in the URL	Binary	Presence of .xml in the URL
	U.15	Mp4a-latm file in the URL	Binary	Presence of .latm in the URL
	D.1	**WWW subdomain in the URL**	**Binary**	**Absence of the WWW subdomain**
	D.2	Subdomain	Binary	Presence of subdomain
	D.3	Suspicious TLD	Binary	Presence of suspicious TLD (based on the list) in the

Feature Category	#	Feature name	Type	Description
				URL
	D.4	**Domain/Subdomain entropy**	**Real**	**Max entropy of domain/subdomain**
	D.5	**Domain/subdomain with uncorrelated words**	**Binary**	**Presence of domain/subdomain with uncorrelated words**
	D.6	**Domain/subdomain length**	**Integer**	**The max length of domains/subdomain**
	D.7	**Suspicious domain name**	**Binary**	**Presence of a suspicious domain name**
Connection Features	C.1	Number of HTTP GET method	Integer	The number of GET requests that appear in the network activity
	C.2	The number of HTTP POST method	Integer	The number of POST requests that appear in the network activity
	C.3	The average length of HTTP response	Real	The average length of the HTTP response in the network activity

4.5 Exploit Kits Network Activity Detection Overview

Exploit Kits Network Activity Detection (EKnad) proposes an ML-based detection methodology as depicted in Figure 4-2. This methodology contains two phases: i) the pre-processing phase that contains two steps, the HTTP Traffic Extraction and the creation of potential EK sessions, and ii) the EK detection phase that comprises three steps: feature extraction, feature selection, and classification. Next, the details of each phase and steps are elaborated.

4.5.1 Pre-processing

4.5.1.1 HTTP Traffic Extraction

The HTTP traffic extraction is the process that extracts only the information related to the HTTP protocol from the PCAP file since the rest of the protocols do not provide any valuable information to facilitate EK detection. For example, the DNS protocol is removed because domains are not used by EKs, since typically they redirect directly to IP addresses. In this step, all the HTTP flows (i.e., HTTP requests and response packets) are extracted and saved in separate text files. By discarding unnecessary information from the rest of the protocols, the classifiers' training time was reduced.

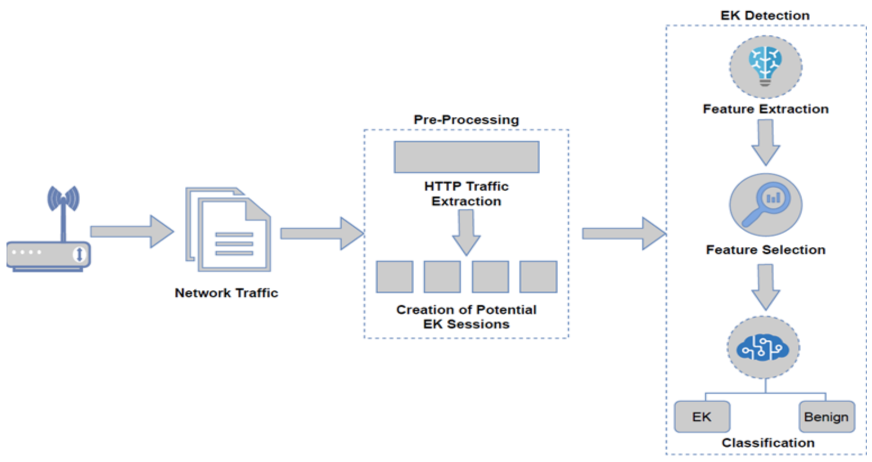


Figure 4-2 EKnad Overview

4.5.1.2 *Creation of Potential EK Sessions*

Modern websites often link to dozens of sites from a single site, so the number of web pages accessed from a web browser ranges from thousands to tens of thousands. This means that the number of HTTP flows to be checked are very large and thus, it is difficult to analyze them directly. For this reason, we group all the HTTP requests and responses of an HTTP session from a specific domain and all the redirection chains that were generated by this domain (if any) as one single session named Potential EK session. A potential EK session resembles an HTTP (or Web) session, but the former captures the network traffic of an attack pattern of an EK, while the latter is the network traffic generated by a particular TCP connection established for example when visiting a website on the Internet. The rationale behind this grouping is to isolate potentially malicious EK traffic and extract features on the basis of each potential EK session. In this way, the HTTP traffic is divided into smaller parts to improve the detection accuracy and noise coming from lower network layers which are all captured in a packet trace (e.g., IP layer headers, layer 2, etc) is removed.

Algorithm 1 depicts the process of generating potential EK sessions. The input of the algorithm is the HTTP traffic that has been extracted from the network traffic in the previous step (i.e., HTTP Traffic Extraction) and the output is a set of potential EK sessions. The main functions of the algorithm are the identification and extraction of: (1) the traffic that is generated by a specific domain and (2) the redirects that have been occurred. The identification of the HTTP traffic of a specific domain is achieved by inspecting the Host HTTP header (starting from the first domain that appears in the HTTP traffic) of successive HTTP requests in a loop, which ends when a new domain name (or IP address) appears in the HTTP traffic (lines 22-33). Furthermore, the HTTP requests that do not have a response are discarded. Particularly, when a new request is processed, algorithm 1 checks whether this request has a response (lines 8-12 and 26-32). In case that there is no response, the potential EK session ends, and the algorithm proceeds to the next request. The redirects are discovered in 4 ways: a) by comparing the domain in the Referer HTTP header with the domain of the host of the previous HTTP request (lines 37-39), b) by comparing the domain of the Host HTTP header with the domain of the Location HTTP header of the previous HTTP response (lines 40-42), c) by checking whether the code of the HTTP response refers to a redirect code (e.g., 300, 301, 302, etc.) and concurrently whether the Location HTTP header of the response contains the same domain as the host of the previous request (lines 43-45), and d) by checking if the domain of the Host HTTP header appears in the body of the previous HTTP response, to capture cases such as redirects via iframes (lines 46-48). In case that there are no further redirects, the potential EK session is finalized (lines 49-51), and the algorithm continues with the next visited domain in the HTTP traffic. The process is repeated until there are no other domains in the HTTP traffic.

Algorithm 1 has been designed to be deployed into individual hosts. In order to operate at a network level, the HTTP traffic should be captured at the entry of (sub)networks. In this case, algorithm 1 should first separate the network traffic based on the IP addresses of the individual hosts and then create potential EK sessions for each host. Capturing the HTTP traffic of a (sub)network requires either a special hardware device namely Network Test Access Point (TAP) or a router with a SPAN port to mirror the traffic. In general, we argue that the deployment of algorithm 1 into a larger network depends on the considered network architecture.

Algorithm 1: Generating Potential EK sessions

```
Input HTTP Traffic;
counterA ← 0;
counterB ← 0;
counterC ← 0;
redirectCode ← [300, 301, 302, 303, 304, 307, 308];
hostHeader [counterA] ← domain_host_header;
potential_EK_session[counterB] ← http_request;
if http_response is EMPTY then
    exit;
else
    potential_EK_session[counterB+1] ← http_response;
end
startSession ← True;
while startSession = True do
    Continue to next HTTP Request;
    if http_response is EMPTY then
        exit;
    else
        counterA ← counterA+1;
        counterB ← counterB+2;
        hostHeader[counterA] ← domain_host_header;
        while hostHeader[counterA] = hostHeader[counterA-1] do
            potential_EK_session[counterB] ← http_request;
            potential_EK_session[counterB+1] ← http_response;
            Continue to next HTTP Request;
            if http_response is EMPTY then
                exit;
            else
                counterA ← counterA+1;
                counterB ← counterB+2;
                hostHeader[counterA] ← domain_host_header;
            end
        end
        refererHeader ← domain_referer_header;
        responseCode ← http_response_code;
        locationHeader[counterC] ← domain_location_header;
        if refererHeader = hostHeader[counterA-1] then
            potential_EK_session[counterB] ← http_request;
            potential_EK_session[counterB+1] ← http_response;
        else if hostHeader[counterA] = locationHeader[counterC-1] then
            potential_EK_session[counterB] ← http_request;
            potential_EK_session[counterB+1] ← http_response;
        else if responseCode = redirectCode AND locationHeader[counterC] = hostHeader[counterA]
          then
            potential_EK_session[counterB] ← http_request;
            potential_EK_session[counterB+1] ← http_response;
        else if hostHeader[counterA] in http_response.body[counterA-1] then
            potential_EK_session[counterB] ← http_request;
            potential_EK_session[counterB+1] ← http_response;
        else
            startSession ← FALSE;
        end
        counterC ← counterC+1;
    end
end
Output Potential EK sessions;
```

4.5.2 EK Detection

4.5.2.1 Feature Extraction

As its name implies, the feature extraction step in EKnad extracts the features from the potential EK sessions along with their numerical values. To this end, we built a joint set F that contains all the observable feature categories of the taxonomy:

$$F = F_{Header} \cup F_{Redirection} \cup F_{Evasion} \cup F_{URL\&Dmain} \cup F_{Connection} \qquad (11)$$

The feature taxonomy that comprised the feature set F is:

$$F_{Header} = \{F_{H.CT.1}, F_{H.CT.2}, \dots, F_{H.CT.19,} F_{H.UA}, F_{H.CD}, F_{H.CE}\}$$

$$F_{Redirection} = \{F_{R1}, F_{R2}, F_{R3}, F_{R4}\}$$

$$F_{Evasion} = \{F_{E1}, F_{E2}\}$$

$$F_{URL\&Domain} = \{F_{UD.1}, F_{UD.2}, F_{UD.3}, \dots F_{UD.22}\}$$

$$F_{Connection} = \{F_{C1}, F_{C2}, F_{C3}\}$$

Utilizing the set F, we define an |F|-dimensional vector space that is represented by binary values, integers, and real numbers. Particularly, 36 features are represented by binary values, nine features are integer numbers, and two features are real numbers. The value of each feature is shown in Table 13. Of note, the features were extracted from the HTTP traffic by implementing a Python program, which takes as input the traffic of the potential EK sessions and generates a Comma-Separated Value (CSV) file that contains the values of the features.

4.5.2.2 Feature Selection

Feature Selection is an essential data processing step to identify the most important features prior to applying an ML algorithm. Its objective is to remove irrelevant or redundant features without much loss of information. The advantage of Feature Selection is three-fold: a) To achieve faster training of the algorithm by reducing the number of features, b) to avoid overfitting the algorithm by deploying only the required features, and c) to attain better prediction results (i.e., improve the classification accuracy) by employing only the most informative features. The Feature Selection methods can be broadly grouped into four categories, named filter, wrapper, embedded, and hybrid methods [155]. The filter method evaluates the features according to heuristics based on general characteristics of the training data, without relying on any learning algorithm. In contrast, the wrapper, embedded, and hybrid methods require a learning algorithm and rely on its performance to determine the effectiveness of the features. To this end, the latter methods are insufficient for EKnad, as they contradict the generalization scope of the proposed methodology. Hence, to identify the most relevant features of EKs' HTTP activity, we employed a filter-based method. We chose the Information Gain (IG) feature selection technique [156] as it has been proved to be very efficient in EK detection field [92], [99].

The IG (or entropy) is calculated for each feature for the output variable, which in our case is the class that the network activity belongs to, namely EK or benign. The IG is defined as the reduction in the uncertainty about the class C of an item having the feature F. The uncertainty is measured in entropy and for C is defined as:

$$H(C) = -\sum \quad P(c_i)log_2(P(c_i)) \qquad (12)$$

where *P(c$_i$)* is the prior probabilities for all values of *C*. The uncertainty for *C* after observing *F* is determined by the conditional entropy defined as:

$$H(C|F) = -\sum_j \quad P(f_j)\sum_i \quad P(c_i|f_j)log_2(P(c_i|f_j)) \qquad (13)$$

where *P(c$_i$| f$_j$)* is the posterior probabilities of *C* given the values of *F*. The IG of a feature *F* with respect to the class *C*, is its mutual information with respect to *C* and is defined as:

$$IG(F) = H(C) - H(C|F) \qquad (14)$$

Respectfully to the IG, it is clear that a feature *F* is considered more correlated to class *C* than feature *X*, if *IG(C|F) >IG(C|X)*. The feature ranking can be calculated by equation (14). Using the equation (14) we calculated the IG value for each feature and ranked them based on their values in descending order. The IG values ranged between 0.3135 and 0.00014. Next, we performed multiple experiments with different selection of features based on their IG values. Specifically, we selected: i) features with above 0.2 IG value (5 features in total), ii) features with above 0.1 IG value (9 features in total), iii) features with above 0.04 IG value (14 features), iv) features with above 0.02 IG value (20 features), v) features with above 0.01 IG value (26 features), and vi) features with an IG value above 0.00014 (all 47 features). For all ML algorithms except MLP, the best detection results were achieved when we selected the features with above 0.02 IG value, which were 20 in total. The results of the MLP algorithm were not affected significantly and the same detection performance was achieved when used 20, 26, and 47 features. Thus, in order to attain the best performance with the minimum overhead, 20 features were selected. The selected features are presented with bold font in Table 13.

4.5.2.3 Classification

The classification step is the crux of the proposed methodology. It is a binary classification task, where the deployed EK samples were acquired from publicly available sources, namely the class of the EK samples is already known. For this reason, we have utilized supervised learning, while unsupervised learning is more appropriate for clustering unlabeled data (i.e., grouping the data based on similarities or differences without prior knowledge of their class) [157]. Furthermore, during the evaluation process, we tested a DL approach, where the MLP ANN was applied with various hidden layers (i.e., 1, 2, 3, and 4). However, the performance improvement with more than two hidden layers was not significant, hence, there was no need for employing a DL approach. In supervised learning, in the learning/training phase, there are input variables x and output variables Y and the deployed algorithm learns the mapping function f to predict the output based on the input $Y = f(x)$. In this research, the input x of the supervised learning refers to the network activity (i.e., the HTTP traffic), and the output Y depicts the class of the network activity (EK or benign).

A challenge of this research is to identify the most appropriate ML algorithms to classify network traffic to EK or benign. To this end, we considered several supervised ML classifiers from various categories: 1) the ensemble learning Random Forest (RF) [158], 2) the probability-based Bayesian Network (BN) [159], 3) the rule-based Decision Table (DT) [43], 4) the Artificial Neural Network (ANN) Multilayer Perceptron (MLP), 5) the instance-based k-Nearest Neighbors (KNN), and 6) the tree-based J48. The aforementioned algorithms that have been extensively utilized in the malware detection field for classification purposes [33], [160] and they will be also deployed in the classification step of EKnad. The classification is divided into two phases; (1) the training phase before application, where the algorithm is trained on a dataset, and (2) the testing phase, where the model is applied to unknown data (data that has not been used in the training phase) to make predictions. An essential task of the learning phase is hyperparameter tuning. Hyperparameters are the parameters of the algorithms that are deployed to control the learning process. In this research, the estimation of the hyperparameters' values is performed via manual tuning. The latter may require more effort, but it facilitates the understanding of the behavior of each algorithm and how the alterations of the hyperparameters affect the learning phase of the algorithm.

4.6 Experimental Evaluation

4.6.1 Evaluation Approach

To assess the detection capabilities of EKnad thoroughly and correctly, state-of-the-art rules and guidelines for evaluating ML algorithms in the computer security domain were considered. More specifically, there is a significant body of works (e.g., [161], [162]) that have identified pitfalls in the evaluation procedure of ML algorithms when they are applied to detect malware, spam and network attacks. During this thesis it has been discovered that the same erroneous evaluation methodologies have been also carried out in previous works related to EK detection, undermining the trustworthiness of their obtained experimental results.

In the following, the most relevant rules are presented that should be followed for evaluating the performance of an EK classifier, in order to avoid introducing bias and attain misleading results.

1. A1: In real world scenarios, malware is encountered much less frequently to goodware. Accordingly, the testing dataset must reflect this observation and utilize a realistic malware-to-goodware ratio [161]. In the case of this thesis, EKs are the minority class, while the benign traffic is the majority class. Thus, an imbalanced dataset must be used in which the EKs samples are far less than the benign traffic. Violation of this rule, means that balanced datasets are used and numerical results may be inflated. A previous work that has not considered this rule is [53] that utilized a testing dataset with 9,362 EKs and 10,541 benign samples, which accounts for almost 50% EK and 50% benign samples.
2. A2: Samples from the same malware family should not be used in both training and testing steps. As malware samples from the same family bear similarities, it becomes evident that violating this rule may boost the performance of the classifier. This rule is violated when the evaluation of the classifiers is based only on the K-fold cross-validation, since the latter uses all the available samples for both training and testing. This rule applies also in EK detection, since various EKs have enough "code and functionality overlap" to be considered as part of

the same family. Many past papers for EK detection do not take into account this rule. For example, the work in [88] utilize only K-fold cross-validation to estimate the accuracy of the EK detection classifier.

3. A3: The training dataset must be temporally precedent to the testing ones. Otherwise, results may be inflated by including future knowledge in the classifier. An example of a past work in the field of EK detection that violates this rule is [92].
4. A4: The ML based detection system must be evaluated using more than one classifier. Although this may seem evident, there are some works for EK detection such as [52] that consider only one classification algorithm to obtain results.
5. A5: Benign and malware samples should not correspond to different time periods [161]. Otherwise, the classifier Defences Against Adversarial Machine Learning Attacks may learn to distinguish applications from different time periods, due to feature differences coming from time decay leading to artificially high performance. For instance, old malware samples may use an old (or even deprecated) version of an API and new benign samples may use an updated version of the API with different parameters or naming. The updated API may be sufficient for the classifier to distinguish malware from benign samples. The past works in EK detection provide incomplete or vague information regarding their experiments, hence it is whether they violate this rule.

From the above rules, EKnad readily complies with Rule A4 and with Rule A5. More specifically, EKnad will utilize not one but six different classifiers to select the most efficient one (i.e., RF, BN, DT, KNN, J48, and MLP). Regarding Rule A5, EKnad is based on stable HTTP features (i.e., HTTP headers), which have not been modified over the years. Therefore, an argument of this thesis is that EKnad is not affected by the different time periods that the benign and EK traffic has been collected. In the next section, it is discussed the dataset creation and the experiments orchestration to abide by the remaining rules (i.e., A1-A3).

4.6.2 Datasets and Experiments

The EK network activity files (in PCAP file format) were collected from the *malware–traffic-analysis.net* website, which contains several EK PCAP files from 2013 until the time of writing this paper (2021). All the PCAP files were download from this website (more than 850) and it was performed a preliminary analysis in order to remove PCAP files that included the same EK. As such, 114 unique EK samples (i.e., PCAP files) were obtained from 2013 to 2020. The collected EK files belong to 26 different EK families, including well-known EKs such as Angler, Rig, Blackhole, Neutrino, Nuclear, Magnitude, as depicted in Table 14. The utilized EK network traffic covers all the state-of-the-art exploits and malwares that EKs have used over the years 2013-2020. Particularly, the EK samples exploit several vulnerabilities, for instance, CVE-2017-8570, CVE-2017-0143, CVE-2018-11776, CVE-2016-4117, CVE-2018-4878, and CVE-2020-0674. The majority of these vulnerabilities belong to the Internet Explorer, Microsoft Edge, Mozilla Firefox, and Google Chrome browsers, Adobe Flash and Silverlight plugins. Furthermore, the EK samples contain the network traffic of the malware delivery process from various malware families, such as RATs (njRAT, Quasar, Gh0st, etc.), Trojans (Smokeloader, Andromeda, AZORult, Zeus, etc.), Botnets (Mirai, Zbot, DanaBot, etc.), Ransomwares (Eris, Maze, CryptoWall, Magniber, etc.), and Coinminers. Hence, the collected dataset provides a thorough depiction of the EK activity in the wild. Regarding the benign samples, to the best of our knowledge there are no public datasets of benign network traffic, thus, the benign network traffic was acquired from the laboratory of the University of Piraeus. The traffic was captured during the period December 2020 to January 2021 using Wireshark. The network traffic originated from ten different desktop computers and resulted in 468 PCAP files. Next, these PCAP files were parsed on the VirusTotal API to check whether they will be flagged by AV products. From this test, 24 PCAP files were excluded leaving in total 444 benign network traffic files.

The EK and benign samples were combined to build five distinct datasets named: i) all-EK dataset, ii) old-EK dataset, iii) new-EK dataset, iv) popular-EK dataset, and v) unknown-EK dataset. To abide by Rule A1 (i.e., the EK network traffic should be much less than the benign), the ratio between EK and benign samples is approximately 25% in all datasets. The popular-EK dataset is comprised by 65 EK samples from the most well-known EK families (RIG, Neutrino, Blackhole, Magnitude, Angler, and Nuclear) and 249 randomly selected benign samples, while the unknown-EK dataset includes 49 EK samples from the remaining 20 EK families that are less known and 195 randomly selected benign samples. The old-EK dataset is comprised of EK samples collected from the years 2013-2016, which contains 74 EK samples from 21 EK families, and 286 randomly selected benign samples. The new-EK dataset is comprised of EK samples from the years 2017-2020, which contains 40 EKs from 9 EK families and 158 randomly selected benign samples. The all-EK contains all the EK and benign samples, namely 114 EK and 444 benign samples. Please note that techniques particularly designed for imbalanced data were not considered, since the main objective is not to address the data imbalance issue, but to validate the detection capabilities of EKnad.

To thoroughly evaluate EKnad four distinct experiments were performed. In **experiment 1**, it was tested whether EKnad can detect less-known EK families, when it is trained on well-known EK families. As such, it was deployed the popular-EK dataset for training the classifiers and the unknown-EK dataset for testing them. In this way, it was guaranteed the fact that the same EK family will not be deployed in both training and testing phases (compliance with Rule A2). In **experiment 2**, it was investigated if EKnad can detect state-of-the-art EKs, when trained on a dataset of older EKs. Particularly, the EK samples has been separated based on the chronological capturing of the EK network traffic. The rationale here is that new EK is not used for the training of the classifiers; instead, training is based exclusively on old EKs, while testing is based on new EKs (compliance with Rule A3). As such the old-EK dataset was utilized for training the classifiers and the new-EK dataset for testing them. In the **experiment 3**, it was deployed K-fold cross-validation in the all-EK dataset. Although the K-fold cross-validation is not sufficient enough to accurately depict the models performance (see Section 4.6.1 - Rule A2), it is one of the most common evaluation methods of classifiers in the literature. Thus, experiment 3 complements the first two experiments and allows us to have a holistic view of EKnad's efficacy. Finally, in **experiment 4**, it was compared the detection rate of EKnad with two prominent open-source IDS named Snort (version 2.9)[30] and Suricata (version 5.0.3)[31] using the old-EK and the new-EK datasets. It should be reiterated the fact that all experiments comply with Rule A4, since we utilize multiple classifiers for the evaluation of EKnad and also with Rule A5, since we do not utilize any time-affected feature that would help the classifiers to distinguish EK traffic from benign traffic. To facilitate researchers, reproduce the results of the experiments and compare EKnad with future solutions the deployed datasets are publicly shared [163].

Table 14 EK Families

#	EK	#Samples	#	EK	#Samples
1	Neutrino	6	14	Zuponcic	1
2	DotkaChef	2	15	FlashPack	4
3	Cool	1	16	Goon/Infinity	4
4	Terror	3	17	Nebula	1
5	g01pack	1	18	Nuclear	6
6	Blackhole	2	19	Angler	6
7	Grandsoft	2	20	Magnitude	6
8	SweetOrange	3	21	Fiesta	8
9	Sibhost	1	22	Rig	39
10	Gondad	1	23	KaiXin	4

[30] https://www.snort.org/

[31] https://suricata-ids.org

11	Whitehole	1	24	Spelevo	6
12	Sundown	4	25	Fallout	1
13	Styx	1	26	Lord	1
Total					**114**

4.6.3 Metrics

For the experimental evaluation of EKnad it was deployed the *recall, precision, FPR, FNR, accuracy, F1-score, MCC, and AUC* metrics that discussed in Chapter 3, Section 3.4.3 have been deployed. In addition, the *Detection Rate (DR)* metric was also used to compare the performance of EKnad with state-of-the-art IDSs (experiment 4). DR is defined as the number of successfully detected EKs divided by the number of all tested EKs.

4.6.4 Results

4.6.4.1 First Experiment: EK family separation

The goal of this experiment is two-fold. First, to deploy different EK families in the training phase of EKnad than in the testing. Second, to asses whether EKnad can detect less-known EKs, when it is trained on well-known EKs (for which there are more available sources). The EK samples were separated based on the EK family they belong. That is, the popular-EK dataset contains EKs from the dominant EK families RIG, Neutrino, Blackhole, Magnitude, Angler, and Nuclear and it was used to train the classifiers, while the unknown-EK dataset is comprised by 20 EK families that are less-known, and it was used for testing the classifiers. Furthermore, the datasets are imbalanced (25% EK over benign samples) to represent a realistic scenario. The results of the first experiment are shown in Table 15. MLP yielded the best performance, since it misclassified only our FNs, which led to 0.983 accuracy, 0.065 FPR, 0.983 F1-Score, and 0.949 MCC. RF misclassified eight FNs and two FPs resulted in 0.959 accuracy, 0.133 FPR, 0.869 MCC, and 0.958 F1-Score, while KNN misclassified eleven FNs achieving 0.954 accuracy, 0.179 FPR, 0.953 F1-Score, and 0.857 MCC. BN also attained comparable results, namely 0.95 accuracy, 0.18 FPR, and 0.949 F1-Score, and 0.842 MCC. J48 and DT accomplished the worst results, misclassifying seventeen and eighteen FNs, as well as three and one FPs respectively. J48 attained 0.918 accuracy, 0.28 FPR, 0.913 F1-Score, and 0.729 MCC while DT achieved 0.897 accuracy, 0.301 FPR, 0.892 F1-Score, and 0.659 MCC. Overall, the best and worst recall, precision, and AUC values were attained by MLP and DT respectively.

The misclassifications of the EK samples are caused because some EKs of the popular-EK dataset, which was utilized for training, deploy different techniques for the exploitation and the malware delivery than the EKs in the unknown-EK dataset that was used for testing. Thus, the values of some features of these two datasets differ. More specifically, the features Content-Type: application/octet-stream and Content-Type: application/ java-archive mostly appear in popular-EK datasets, while the features Content-Type: application/x-shockwave-flash and Content-Type: application/x-msdownload mostly appear in the unknown-EK dataset leading to the erroneous classification of some EK samples.

Regarding the misclassifications of the benign samples, it occurred in samples that have similar features with the EKs. Particularly, the features: Jar file in the URL, Suspicious filename in the URL, high Domain/Subdomain Entropy, Content-Type: application/octet-stream, and Content-Encoding appear both in EK and benign samples. Hence the classifiers could not distinguish the benign network activity from the EK. These misclassifications are difficult to be solved, because even though we have deployed a thorough feature set that considers the whole infection procedure of EKs, there are subtle differences between well-known and unknown EK families that the ML algorithms cannot identify.

Table 15 Results of Experiment 1

Model	Recall	FPR	Precision	Accuracy	F1-Score	MCC	AUC
RF	0.959	0.133	0.959	0.959	0.958	0.869	0.98
BN	0.951	0.18	0.952	0.95	0.949	0.842	0.994

Model	Recall	FPR	Precision	Accuracy	F1-Score	MCC	AUC
DT	0.898	0.301	0.893	0.897	0.892	0.659	0.908
MLP	**0.984**	**0.065**	**0.984**	**0.983**	**0.983**	**0.949**	**0.996**
KNN	0.955	0.179	0.957	0.954	0.953	0.857	0.891
J48	0.918	0.28	0.918	0.918	0.913	0.729	0.928

4.6.4.2 Second Experiment: Chronological separation of EK samples

The goal of the second experiment is to test the detection performance of EKnad on newer EKs, when it is trained on older EKs. As it was mentioned in Section 4.6.2, the EK samples were separated chronologically on two datasets, where the old-EK dataset contains EKs from 2013-2016 and it was used as a training dataset of the classifiers, while the new-EK dataset is comprised by newer EKs from 2017-2020 and it was used for testing the classifiers. As previously, the datasets are imbalanced (25% EKs over benign) to represent a realistic scenario.

Table 16 illustrates the results of the second experiment. As in the first experiment, the MLP achieved the best detection performance, namely 0.989 accuracy, 0.04 FPR, 0.99 F1-Score, and 0.969 MCC, which was the result of only two FNs. Following, RF and BN had the same performance. Particularly, they both attained 0.979 accuracy, 0.08 FPR 0.979 F1-Score, and 0.937 MCC, due to the erroneous classification of 4 FNs. KNN misclassified in total six FNs and attained 0.969 accuracy, 0.12 FPR, 0.969 F1-Score, and 0.905 MCC, while J48 yielded 0.949 accuracy, 0.162 FPR, 0.948 F1-Score, and 0.838 MCC, since it misclassified eight FNs and two FPs. DT attained the worst results in this experiment, as it misclassified twelve FNs and four FPs caused 0.919 accuracy, 0.245 FPR, 0.916 F1-Score, and 0.736 MCC. Overall, MLP had the best recall (0.99), precision (0.99), and AUC (0.999), while the worst recall (0.919), precision (0.917), and AUC (0.97) attained by DT.

Elaborating more on the misclassifications, when it comes to the EK traffic, several differences can be found between the EK traffic of older (i.e., the training samples) and newer EKs (i.e., the testing samples), which are reflected on the features. For instance, the majority of older EKs utilized IP addresses instead of domain names, while newer EKs use Domain/Sub-domain with uncorrelated words or Suspicious domain names. These differences are reflected in the URL & Domain features, which potentially caused the misclassifications of the EK samples. Regarding the misclassifications of the benign traffic, the reasons are similar with the previous experiment, namely the benign samples included features that appeared also in the EK samples.

Table 16 Results of Experiment 2

Model	Recall	FPR	Precision	Accuracy	F1-Score	MCC	AUC
RF	0.98	0.08	0.98	0.979	0.979	0.937	0.98
BN	0.98	0.08	0.98	0.979	0.979	0.937	0.98
DT	0.919	0.245	0.917	0.919	0.916	0.736	0.97
MLP	**0.99**	**0.04**	**0.99**	**0.989**	**0.99**	**0.969**	**0.997**
KNN	0.97	0.12	0.971	0.969	0.969	0.905	0.925
J48	0.949	0.162	0.949	0.949	0.948	0.838	0.896

4.6.4.3 Third Experiment: 10-fold Cross-Validation

The third experiment aims on the evaluation of EKnad on all the EK samples (i.e., the all-EK dataset) using the K-fold cross validation with K=10. In K-fold cross-validation [164] the dataset D is randomly split into K mutually partitions (or folds) D1, D2. . . DK of approximately equal size. Then, the algorithm is trained and tested K times, where each time a different fold of data is maintained for testing, and the remaining K − 1 folds are used for training. Every time the evaluation metrics were measured and then, a rotation takes place to change the fold that will be maintained for testing, and consequently, the remaining folds will be deployed for training. The K-fold cross-validation ends by calculating the mean value of all rotations for all the metrics. The advantage of this method is that all instances in the dataset are eventually used for both training and testing purposes.

The performance results of the 10-fold cross-validation on MLP, RF, BN, DT, KNN and J48 classifiers are depicted in Table 17. We observed once again that MLP outperformed all classifiers, since it accomplished 0.991 accuracy, 0.028 FPR, 0.991 F1-Score, 0.972 MCC. The results occurred after MLP mis-classified four FNs and one FP (benign classified as EK). Following, the RF classifier attained 0.989 accuracy, 0.042 FPR, 0.989 F1-score, 0.967 MCC, which was the result of the erroneous classification of six FNs. BN misclassified seven samples in total; five FNs and two FP, accomplishing 0.987 accuracy, 0.036 FPR, 0.987 F1-Score, 0.961 MCC. KNN attained 0.983 accuracy, 0.063 FPR, 0.984 F1-Score, 0.95 MCC, as a result of the misclassification of nine FNs. The tree-based classifier J48 misclassified fourteen FNs and nine FPs, achieving 0.958 accuracy, 0.102 FPR, 0.958 F1-Score, 0.871 MCC. Finally, DT accomplished slightly worst results in detection accuracy (0.955), F1-Score (0.956), and MCC (0.867) than J48, however, it accomplished better FPR (0.064). Particularly, it failed to classify eight FNs and seventeen FPs. Regarding the remaining evaluation metrics, MLP achieved the best results in recall (0.991) and precision (0.991), while DT yielded the worst results on recall (0.955) and precision (0.957). The MLP along with RF and BN achieved the best AUC value (0.999). J48 had the lowest AUC value (0.929), which was worse than DT (0.976) and KNN (0.965). It can be assumed that the misclassifications of the EK samples are caused because the network traffic of these EKs present similarities at a domain level with the benign traffic. That is, the values of some URL & Domain features, such as Domain/Subdomain Length, Domain/Subdomain entropy, Max no. of “-” in the URL, and Max no. of “/” in the URL, were close to benign samples.

Table 17 Results Experiment 3

Model	Recall	FPR	Precision	Accuracy	F1-Score	MCC	AUC
RF	0.989	0.042	0.989	0.989	0.989	0.967	0.999
BN	0.987	0.036	0.987	0.987	0.987	0.961	0.999
DT	0.955	0.064	0.957	0.955	0.956	0.867	0.976
MLP	**0.991**	**0.028**	**0.991**	**0.991**	**0.991**	**0.972**	**0.999**
KNN	0.984	0.063	0.984	0.983	0.984	0.95	0.965
J48	0.959	0.102	0.958	0.958	0.958	0.871	0.929

4.6.4.4 Fourth Experiment: Comparison with Intrusion Detection Systems

The objective of the last experiment is to test whether EKnad is more efficient in the detection of EKs’ network traffic than two well-known rule-based IDS. In particular, the DR and the FPR of Snort and Suricata IDS were evaluated against EK network traffic and compared the results with EKnad using the MLP classifier (since it achieved the best results in the previous three experiments). The old-EK and the new-EK datasets were utilized to investigate whether the DR of the IDSs deteriorates on newer EKs. To compare the results of IDSs with EKnad, 10-fold cross-validation was deployed in the old-EK dataset to obtain numerical results, while the same numerical results of the previous experiment were utilized for the new-EK datasets. It is evident that the detection performance of rule-based IDSs is directly related to the deployed rulesets. If the related rule for the detection of a specific EK sample is available, then the IDS will be able to detect and trigger an alarm. Snort and Suricata come with pre-installed rules, but these are limited and not sufficient to detect EKs. To this end, for Snort the EK rules from the latest subscribers rules (at the time of writing it was the snapshot 29838) were deployed. Regarding Suricata, it was used the Proofpoint Emerging Threat (ET) open ruleset[32]. In this way, the deployed ruleset of Snort contained 751 EK rules, while the ruleset of Suricata was

[32] https://rules.emergingthreats.net/OPEN download instructions.html

comprised of 1,132 EK rules. It is important to mention that the ET open ruleset also provides rules for Snort; however, this ruleset was not deployed, because it did not contain a specific EK rule category. Before presenting the results of this experiment, it should be clarified that the tested IDSs were installed only with the obtained EK-specific rules to ensure that the IDSs will trigger an alarm only due to the detection of EK activity and not from other suspicious activities (e.g., half-open TCP connections, blacklisted URL domains, etc).

Results show that EKnad detected 72 out of 74 EKs in the old-EK dataset, accomplishing a 97.29% DR of the EK samples. Snort detected in total 65 EKs and its DR was 87.83%, while Suricata yielded worse results than Snort. Particularly, Suricata detected 58 EKs, which means a 78.37% DR. Moreover, in the new-EK dataset, EKnad detected 39 out of 40 EKs, accomplishing a 97.5% DR. In this dataset, Snort achieved worst results than Suricata. More specifically, Snort detected 32 out of 40 EKs, while Suricata detected 37 out of 40, namely Snort and Suricata reached 80% and 92.5% DR respectively. Regarding the FPR results, it is known that rule-based IDS typically have lower FPR than ML-based detection methods. In this experiment, Snort and Suricata attained the same FPR, namely 0.3% in the old-EK dataset and 0.6% in the new-EK dataset, due to the erroneous classification of only one benign sample in both datasets. On the other hand, EKnad misclassified three benign samples resulting in 1% FPR. However, in new-EK dataset it achieved the same FPR with Snort and Suricata, namely 0.3%. The results of this experiment are summarized in Table 18.

Elaborating more on the results, EKnad did not detect two EK samples from the category old-EK dataset and one from the new-EK dataset. In old-EK dataset, Snort did not detect 9 EKs samples from various EK families. Particularly, it did not detect: 1 out of 6 (1/6) Neutrino EKs, 2/4 Sundown EKs, 1/14 Rig EKs, 3/8 Fiesta EKs, 1/6 Magnitude EKs, and 1/1 Gondad EK. For the same dataset, Suricata did not detect 16 EKs samples: 3/3 Neutrino EKs, 1/4 Sundown EKs, 1/14 Rig EKs, 2/8 Fiesta EKs, 1/1 Gondad EK, 4/6 Angler EKs, 2/6 Nuclear EKs, 1/3 Flashpack EKs, and 1/3 SweetOrange EKs. In the new-EK dataset, Snort did not detect 8 EK: 6/24 Rig EKs, 1/1 Lord EK, and 1/1 Fallout EK; while Suricata did not detect 3 EKs, 1/1 Fallout EK, and 2/6 Spelevo EKs. As far as it concerns the potential reasons for the FP results of EKnad in the old-EK dataset, the benign samples, which were erroneously classified as EK, were comprised by features, such as Content-Type: application/octet-stream, Content-Encoding, Jar file in the URL, Suspicious filename in the URL, and high Domain/Subdomain Entropy that typically appear in EK samples. Thus, EKnad in the old-EK dataset achieved higher FPR than Snort and Suricata. The findings of the fourth experiment indicate that EKnad performs better than Snort and Suricata in terms of detection of both old and new EKs. Even though the EK samples deployed in the experiment are publicly available, the EK rulesets are not sufficient enough to detect all EKs. Another interesting finding is that the community ruleset that was deployed in Snort is more robust on old EKs, while the ET open ruleset that deployed in Suricata is more effective on newer EKs. It is important to mention that both IDSs have commercially available rules (that this thesis was not able to validate), which might provide a more robust detection of EKs.

Table 18 Results Experiment 4

Detection Method	**DR old-EK**	**FPR old-EK**	**DR new-EK**	**FPR new-EK**
EKnad (MLP)	**98.64%**	1%	**97.5%**	**0.6%**
Snort	87.83%	**0.3%**	80%	**0.6%**
Suricata	78.37%	**0.3%**	92.5%	**0.6%**

4.6.5 Discussion

Based on the results of the experiments 1, 2, and 3, it has been concluded that MLP, in comparison to RF, BN, DT, KNN, and J48 classifiers, achieved the best detection performance, since it yielded the best results in all experiments. This thesis deduced that MLP, overall, is the most efficient algorithm for the proposed EKnad. Furthermore, the following generic conclusions have been drawn:

5. **MLP:** The MLP classifier has proved to be the most suitable for the detection of EKs network activity. It misclassified the least EK samples in all experiment.
6. **RF:** The RF classifier achieved the second best results in all the experiments.
7. **BN:** The results of BN classifier in the second experiment were similar to RF, while in the first and third experiment its results were slightly worst than RF.
1. **KNN**: The KNN classifier yielded better results in the third experiment than BN (worst than MLP and RF), but in experiments 1 and 2 its results were worst than MLP, RF, and BN.
8. **J48:** The J48 classifier attained slightly better results than DT in all experiments and worst than MLP, RF, BN, and KNN.
9. **DT:** The DT classifier had the worst results of all the tested classifiers in all three experiments.

Regarding the results of the fourth experiment, Snort attained better results on older EKs, whilst Suricata performed better on newer EKs. On the other hand, EKnad with MLP classifier outperformed both Snort's and Suricata's detection capabilities, while accomplishing comparable results in terms of FPs.

Finally, a comparison with previous works is performed as shown in Table 19. In particular, EKnad compared with previous works in terms of common evaluation metrics (e.g., accuracy and precision), detection category, number of EK families, whether they deployed imbalanced data, and whether they publicly share the datasets. As shown in Table 19, the numerical results of EKnad are similar and comparable with the results of previous works. However, even though some works attained high detection performance, the results cannot provide a comprehensive view of how these solutions will perform in real-world scenarios. That is, some previous works (e.g., [88], [89], [90], [53]) deploy balanced datasets, which is not realistic in malware detection. In general, the majority of the related work has not considered the proposed rules of section 4.6.1 (i.e., A1-A5), and therefore, their results may be biased. Moreover, as one can notice in Table 19 some works (e.g., [150]) rely on the source code of EKs to detect malicious activity, which is not robust as obfuscation techniques can hinder detection. Moreover, a higher number of EK families have been considered (26 in total) compared to all other works. Finally, a drawback of the previous works is that they do not share enough information (e.g., datasets). Thus, their results are not reproduceable. The deployed datasets have been publicly shared to drive more research in the EK detection field as well as to assist researchers reproduce the results of EKnad and compare it with future solutions.

Table 19 Comparison with SotA. The "N", "S", and "C" indicate the network traffic, EK server-side, and EK client-side categories respectively

Paper	ACC	Prec.	#EK Families	Imbalanced Data	Data Sharing	Detection Category
Mekky et al.	0.99	0.95-0.99	N/A	✗	✗	N
Taylor et al.	0.95	N/A	6	✗	✗	N
Nikolaev et al.	N/A	0.92-0.95	3	✗	✗	N
Qin et al.	0.95	0.96	N/A	✗	✗	N
Burgess et al.	0.99	0.99	N/A	✓	✗	N
Harnmetta	0.98	N/A	5	✗	✗	N
Eshete et al.	0.99	N/A	11	✓	✗	S
Kim et al.	N/A	0.99	12	✓	✗	C

Yoo et al.	0.98	N/A	11	✓	✗	C
EKnad	0.98-0.99	0.98-0.99	26	✓	✓	N

5. A Novel Platform for Detecting and Exploiting Server-Side JavaScript Injection Attacks

5.1 Introduction

In this thesis, in order to gain a better knowledge about cyberattacks, the efforts focused on creating a platform to automatically detecting vulnerabilities related to a new type of code injection attack, named Server-side JavaScript Injection Attack (SSJI). The rationale for focusing on this particular attack type stems from the fact that recently SSJI-based security gaps on Node.js implementations have been discovered, which discussed in Chapter 1, section 1.1 (i.e., PayPal vulnerability [2]). The SSJI is a powerful attack that can have severe consequences on webservers varying from stealing sensitive information to gaining unauthorized access to remote servers. There are also other similar client-side injection attacks, such as XSS; however, can only execute commands inside the browsers' sandbox, while the novelty of SSJI attacks relies on the fact that they can execute JavaScript code in an unprotected environment, gain access (e.g., via a remote shell) in the server and perform large scale attacks (e.g., APTs). Moreover, compared with other code injection attacks, like SQL injection that have drawn the focus of the research community more than a decade ago [4] [5] [6]; the research on SSJI attacks is in its infancy letting numerous applications unprotected against this threat.

Taking into account the above observations, this chapter will try to answer the following research question:

Is it feasible to create a methodology to automatically detect and exploit SSJI vulnerabilities in Node.js applications? (RQ4).

To answer RQ4, this thesis makes the following contribution:

NodeXP, a Node.js application scanner that can automatically detect and exploit SSJI vulnerabilities in Node.js applications.

More specifically, this thesis proposes NodeXP (NOde.js JavaScript injection vulnerability DEtection and eXPloitation), the first methodology, which was developed as a software tool that automatically detects and exploits SSJI attacks in Node.js applications. The next sections analyze the NodeXP architecture and the approach that was followed for its design and implementation. The vulnerability detection process is based on dynamic analysis, using both result and blind -based injection techniques. Later, the exploitation process begins that aims to establish a remote session with the server automatically. An advantage of NodeXP is that it provides several advanced functionalities, which distinguish it from other relevant tools. A novel aspect of NodeXP is that it obfuscates attack vectors to bypass application-level filtering mechanisms (e.g., blacklisting) as well as network-level mechanisms, like Web Application Firewalls (WAFs) and Intrusion Detection Systems (IDS). Finally, the detection and exploitation capabilities of NodeXP were evaluated on different testing scenarios to assess its effectiveness and compare it with well-known commercial and open-source vulnerability scanners. The experimental results concluded that NodeXP outperformed the well-known vulnerability scanners. As it was mentioned in Chapter 1, section 1.2, NodeXP was released as open-source to facilitate the security community in discovering SSJI vulnerabilities.

The remainder of this Chapter is structured as follows: Section 5.2 presents background information about SSJI and existing works on SSJI attacks and Node.js vulnerabilities. Section 5.3 elaborates on the architecture of NodeXP, while Section 5.4 evaluates its capabilities on several assessment scenarios. Section 5.5 analyzes countermeasures that developers should employ to secure their applications against SSJI attacks. Section 5.6 summarizes the critical points. Please note that parts of this Chapter have been published in [30].

5.2 Background and Related Work

This section aims to provide background knowledge on SSJI attacks to facilitate the reader comprehend the critically of this particular attack type as well as to present the existing literature on Node.js application vulnerabilities and SSJI attacks.

5.2.1 Server Side JavaScript Injection Attacks

SSJI is a code injection attack that occurs in server-side applications that execute JavaScript. Its main aim is to inject arbitrary JavaScript code that will be executed by the application. SSJI vulnerabilities can be found in applications that may insensibly accept user-controllable data, which are dynamically processed by a JavaScript code interpreter, and the developer has neglected to use proper input validation and filtering mechanisms.

SSJI attacks primarily target Node.js applications. This can be attributed not only to the popularity of Node.js as a platform for server-side JavaScript development, but also to its powerful characteristics and wide functionality that create a large attack vector. Indeed, as presented in [34], several Node.js features are easily misused and potentially provoke security vulnerabilities. More specifically, distinct JavaScript functions could be used erroneously and affect Node.js applications. These functions are eval() and Function(). The former takes a string argument and interpret it as JavaScript, allowing an attacker to execute arbitrary code in the context of the current application. The Function() constructor creates functions dynamically and suffers from similar security issues with eval5. It is worth mentioning that Node.js has other dangerous functions such as exec() and its variants that allow execution of operating system commands. If there is no input validation, an attacker can exploit them to inject and execute arbitrary commands. Strictly speaking, this attack known as command injection is considered different from SSJI, since the latter is related to the execution of JavaScript code while the former is related to the execution of system commands. Moreover, contrary to SSJI, command injections depend on the operating system that hosts the web application. Therefore, since command injections comprise a different category of code injections attacks, they are out of the scope of this paper and will not be considered.

The SSJI vulnerabilities can be classified into two main categories: (a) result-based and (b) blind-based. In the result-based SSJI, the response of the injection is displayed on the vulnerable application, and an attacker can directly deduce whether the injected code was executed successfully or not. On the other hand, in blind-based SSJI the output is not displayed on the vulnerable application (i.e., no feedback is received); hence, the attacker cannot directly infer whether the JavaScript code was executed successfully. This means that the attacker must determine whether a web application is vulnerable or not to SSJI through other means, usually using the server's response time to the injection request [35]. Through this technique – also popular in SQL and command injections [6] - an attacker injects and executes JavaScript code that causes a time delay in the response. By measuring the time it took the application to respond, the attacker can identify if the injected code was executed or not.

5.2.2 Server-side JavaScript Injection Attacks' Impact

Compared to other code injection attacks such as XSS and SQL injections, SSJI attacks may not be so prevalent. However, the security consequences of SSJI attacks can be significant and costly. In particular, the impact of SSJI attacks ranges from trivial denial of service to unauthorized remote access to the system that hosts the vulnerable web application [36]. In the latter case, the damage can be substantial as an attacker can gain access to resources that include sensitive data (e.g., passwords), delete files or add new system users for persistence. To achieve the above, an attacker can exploit a set of functions readily available in the Node.js platform. To elaborate more, Node.js introduces to JavaScript the Child Process module, which includes the function exec() and its variants (i.e., spawn(), execfile() and fork(). The above functions allow a Node.js application to access low-level resources of the operating system that is hosting the application. Note that these functions are not available in traditional client-side JavaScript applications. For instance, exec() allows Node.js applications to execute system level commands. Thus, an attacker can inject code that leverages exec(), in order to access low-level resources (e.g., by executing the ps command an attacker can view the running processes). Additionally, Node.js includes the fs module, which includes among others the readFileSync() and writeFileSync() functions to read and write the contents of a file respectively.

5.2.3 Existing Literature on SSJI Attacks and Node.js Vulnerabilities

The literature in the field of SSJI and in general Node.js vulnerabilities is quite sparse. Ojamaa et al. [10] were among the first who pointed out the security issues and pitfalls that hide behind Node.js applications. Towards this direction, Davis et al. [37] examined the event-driven architecture of Node.js and observed that due to its single-threaded model, Node.js exposes its applications to a specific class of denial of service attacks (i.e., algorithmic complexity attacks). To support their findings, they examined a set of relevant and popular npm modules and discovered an abundance of vulnerable code snippets exposing several modules to DoS attacks. An extension of the above work is presented in [38], in which the authors examined regular expression-based denial of service attacks in real-world Node.js web applications. They discovered at least 339 popular web applications, which are vulnerable to this type of denial of service attacks.

Regarding SSJI attacks, a recent work that has the same grounds with NodeXP is Synode [39], which attempts to mitigate injection vulnerabilities in Node.js applications. To accomplish that, during the installation process of a third-party Node.js module, a check is performed to detect and rewrite APIs that appear to be prone to injections. Static analysis of the possible input values that will be passed to the aforementioned APIs is performed; if the static analysis does not yield a definitive result, a dynamic enforcement mechanism that blocks malicious inputs before reaching the APIs is deployed. Similarly, Affogato [40] performs dynamic analysis of Node.js modules and applications. It revolves around the idea of using grey-box taint analysis by detecting flows of data from untrusted sources to security-sensitive sinks. A comparison between Affogato and Synode revealed that the former outperforms the latter in terms of detection capabilities, as Synode exhibits a high number of false positives reducing its accuracy.

In [41] a large-scale study about the runtime behavior of the eval() function in JavaScript applications is presented. Their findings were very interesting as they concluded that the eval() funcation can not be always replaced exclusively by other functions. Note that their research focused only on client-side JavaScript applications. Whether their results can be repeated and validated for server-side Node.js applications remains an open question. The authors of [42] present a black-box technique, which can be applied to a wide variety of code injection attacks (SQL injection, XSS) namely taint inference. The source code is not needed as it operates by intercepting the applications' requests and responses. In [43] an automated detection method to discover a wide variety of vulnerabilities in web applications by analyzing the source code is presented. When a possibly vulnerable piece of code is detected, a symbolic execution takes place to determine whether the specific code segment is susceptible to injection attacks or not.

Code injection attacks have been identified also in HTML5 applications. The authors of [44] presented JS-SAN, a framework that mitigates the effect of JavaScript code injection vulnerabilities in HTML5-based web applications. The experimental results revealed that the performance of JS-SAN, in terms of false negatives and false positives, is better than the related works. Gupta et al. [45] pinpointed that HTML5 applications, which are deployed in the cloud, appear to be vulnerable to JavaScript code injection attacks (e.g. XSS). To this end, they developed a defensive framework for cloud-based HTML5 web applications. In the mobile application domain, where JavaScript is also utilized, Yan et al. [46] proposed the deployment of a hybrid deep learning Network to detect code injection attacks on hybrid HTML5 mobile applications.

A reminiscent of code injection attacks is command injection, where the attacker attempts to execute commands of the underlying operating system (for instance, ping, whoami, etc.). Commix [6] is a command injection tool that offers automated vulnerability detection and exploitation. Several 0-day vulnerabilities were detected with the help of commix, and its value has been widely recognized as it comes preinstalled in many operating systems that focus on cyber-security, as well as in the renowned Kali Linux. Moreover, the works in [47] and [48]have reviewed web application protection techniques, and have classified them according to the properties they rely on to detect and/or prevent malicious behaviours. Those properties can be statistics, policies, the intent of the developed application, whether the outcome after the execution of an application is the predicted one or not, etc.

Finally, a plethora of tools called web application vulnerability scanners (e.g., [49], [50]) are available to the public focusing on the detection of a wide variety of vulnerabilities including SSJI among others. However, these scanners are mostly commercial and there is a limited number of free web application scanners. The majority of these scanners do not provide automation of the exploitation process.

5.3 NodeXP Platform Architecture

5.3.1 Overview

NodeXP is a software tool written in Python that implements a methodology to automate the detection and exploitation of SSJI vulnerabilities in Node.js applications. A beta version of NodeXP has been released as open-source [118] to facilitate security researchers and web developers to discover bugs and vulnerabilities related to SSJI attacks.

NodeXP can be utilized to identify critical SSJI vulnerabilities in a wide variety of web applications that are built with Node.js. The latter is considered to be an important component of the IoT domain, since Node.js is capable of handling dynamically changing data and heavy data flows generated by millions of IoT devices. Apart from scalability, Node.js is easy to integrate with IoT as it has built-in support for IoT protocols such as MQTT and websockets. On top of this, Node.js facilitates IoT development, since npm features a lot of useful IoT modules ready to be consumed by applications. Therefore, this thesis argues that NodeXP can be an important tool to protect server-side IoT data and applications. Another area that Node.js thrives is distributed systems and specifically microservices. The main idea behind a microservice architecture is to create small, scalable and loosely coupled functional pieces of an application, which is contradicted to the traditional, monolithic approach where an application is developed as a whole. The nature and goal of both Node.js and microservices are identical at the core, making both suitable for each other. Together used, they can power highly-scalable applications and handle thousands of concurrent requests without slowing down the system. Thus, NodeXP can be a valuable tool for microservices to enhance their security posture.

5.3.2 Architecture

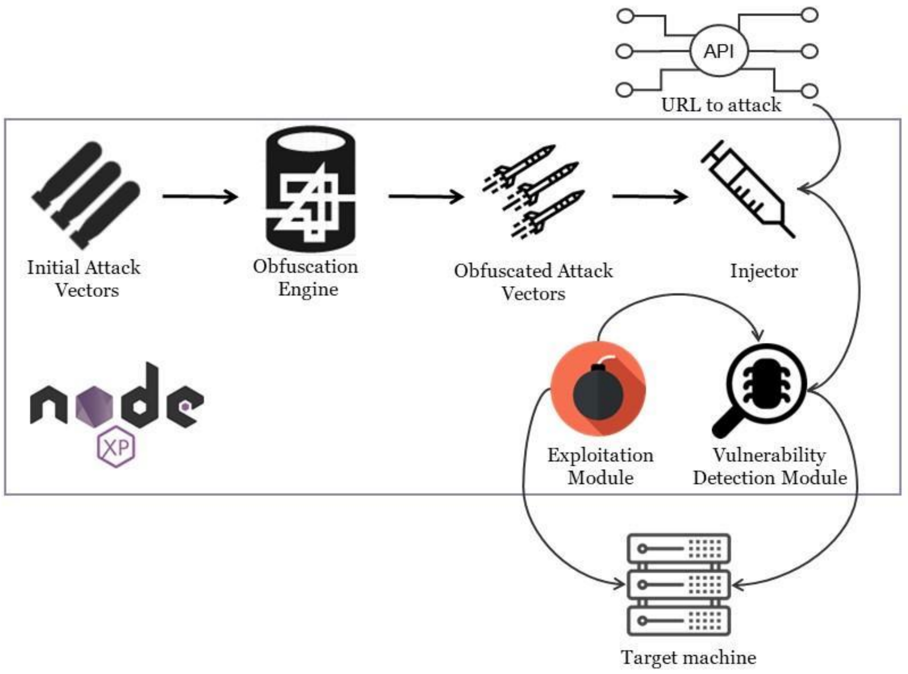


Figure 5-1 NodeXP Architecture

The structure of NodeXP is composed of three main modules: (a) attack vectors, (b) vulnerability detection, and (c) exploitation. Figure 5-1 illustrates the architecture of NodeXP. The modular architecture of NodeXP allows not only the easy addition of a new module, but also bypassing a module in case its functionality is not required (e.g., exploitation may not be the scope of a security testing). This level of flexibility optimizes the process of data handling in NodeXP. The attack vector module manages and processes the attack vectors to be used by the vulnerability detection module. The latter, as its name implies, tries to identify SSJI vulnerabilities based on detection heuristics. Moreover, as it is analysed below, this module tries to minimize false positives by performing several sanity checks. If the vulnerability detection module determines that the application is vulnerable, then NodeXP triggers the exploitation module to attempt automatic exploitation. In the following sections, the vulnerability detection and exploitation module were analysed highlighting their advantageous features.

5.3.3 Attack Vector Module

This module contains the set of attack vectors along with the obfuscation engine. As it is mentioned below, the obfuscation engine is implemented in NodeXP's architecture to bypass security mechanisms on the server-side as well as to increase its detection efficacy regarding SSJI vulnerabilities. An attack vector is comprised of a JavaScript code that performs a pre-determined operation. For instance, a mathematical operation (e.g., the addition of random numbers) parsed by the Function constructor or the eval() function. The underpinning idea is that NodeXP is expecting to get back from the vulnerable application the result of the operation, which is considered a proof that the operation was executed through the SSJI vulnerability. The approach that is followed in NodeXP was to derive a number of well-known attack vectors that are more likely to achieve an SSJI based on our empirical results. The generated attack vectors are passed to the obfuscation engine to encode their contents. Obfuscation is traditionally used by developers, in order to protect their code from reverse engineering. One of the novelties of NodeXP lies in the fact that obfuscation is deployed for a different purpose. In particular, the obfuscation is exploited not to hinder reverse engineering, but to encode the attack vectors. With this way, NodeXP bypasses not only application-level filtering mechanisms (e.g. blacklisting), but also evade network-level mechanisms, like Web Application

Firewalls (WAFs) and Intrusion Detection Systems (IDS). The obfuscation engine of NodeXP deploys the following features:

- Randomization Obfuscation, which replaces the names of variables and functions with randomly created sequences of characters that have no particular meaning.
- Encoding Obfuscation, which converts parts of the code into hexadecimal representations. Particularly, it encodes the variable's values and the inputs that are passed into functions.
- Dead Code Insertion, which adds arbitrary code that executes during the execution of the initial code without affecting its semantics.

For instance, assume a Node.js application that includes a filter that strips the slash character (/) from an untrusted input that will be passed as an argument to the eval() function. Also assume that NodeXP tries to detect the SSJI vulnerability using as an attack vector the code which was presented in Listing 3 that prints the contents of the Linux passwd file. Without obfuscation, NodeXP would not be able to understand whether the application is vulnerable to SSJI. That is, due to the filtering of the slash character, the attack vector will become invalid, and its execution will fail. With obfuscation, the attack vector will be encoded as shown in Figure 5-2. The obfuscated attack vector does not have a slash character and when is passed to eval(), will effectively bypass the application filtering.

```
var _0xf7e5=["\x2F\x65\x74\x63\x2F\x70\x61\x73\x73\x77\x64",
"\x72\x65\x61\x64\x46\x69\x6C\x65\x53\x79\x6E\x63",
"\x66\x73","\x65\x6E\x64"];
res[_0xf7e5[3]](require(_0xf7e5[2])[_0xf7e5[1]](_0xf7e5[0]))
```

Figure 5-2 Obfuscated attack vector

5.3.4 Vulnerability Detection Module

The obfuscated attack vectors are used by the vulnerability detection module. The latter uses two different methods: the result-based and the blind-based detection. Result-based detection uses the attack vectors sequentially and attempts to inject them into possible vulnerable fields (e.g., GET/POST parameters) and evaluate the HTTP response. The rationale is to utilize Node.js functions that execute code and return the result into the HTTP response. To this end, NodeXP records and parses the received HTTP response to verify whether the result is the expected one. Three different cases can be distinguished depending on the outcome of this evaluation. In the first case, the application responds with the expected result of the attack vector's execution (see below). This proves that the application is vulnerable to SSJI. In this case, NodeXP will inform about the discovered vulnerability and the control will be transferred to the exploitation module. In the second case, the application is responding with generic errors or without any errors.

This means that NodeXP is not capable of executing the attack vectors and probably the web application is not vulnerable. As a last resort, NodeXP will try to perform blind-based detection. The third case lies between the previous two cases; the application is not responding with the correct execution of the attack vector, but instead its response contains an error (e.g., HTTP 500 Internal Server Error). This means that the application parses the attack vector but somehow its execution triggered an error message, instead of the desired result. This may lead us to the assumption that the application is likely to be vulnerable; however, there is no conclusive result. In this case, NodeXP continues with the next attack vector; When all the attack vectors of the result-based detection have been used without the expected results, then the tool proceeds with the blind-based detection.

Special attention is given on the aforementioned third case as it is the most important and interesting one. NodeXP attack vectors mainly utilize the Node.js object named "response" to execute code on the server side and read the result of the code execution. Note that eval() can be also used for the same purpose, but it is not considered in this discussion, since web application firewalls or the application itself may block the execution of eval(). One of the barriers that SSJI exploitation faces, is the identification of the correct naming of the response object by Node.js frameworks such as Express or

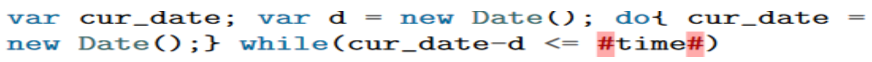
Meteor. More specifically, while Node.js always uses the name “response” to refer to the object that handles server HTTP responses, several Node.js frameworks modify the name of this object. Even worse, many Node.js frameworks allow the arbitrary naming of this object by the developers. A case in point is Express API (one of the most popular Node.js frameworks), which follows the convention of referring to the response object as “res” as opposed to “response”. Moreover, the Express API documentation [119] goes on to make the point that developers do not have to follow this convention. More specifically, as mentioned in [119]: “In this documentation and by convention, the object is always referred to as req (and the HTTP response is res) but its actual name is determined by the parameters to the callback function in which you’re working.”. In other words, the response object naming is arbitrary and could be called anything. Without knowing the correct object name, SSJI attacks cannot be performed successfully.

To overcome this obstacle and improve its overall robustness, the detection engine of NodeXP performs the so-called “enumeration of the response object” to guess its correct naming. First, NodeXP utilizes an attack vector that uses the common object name “response”. If the server responds with a HTTP 500 Internal Server Error due to a Reference Error or Type Error, then NodeXP provides an indication that the attack vector was injected into a Server-Side JavaScript parser, but the name “response” is not the correct name for this object (hence the error message). Thus, NodeXP modifies the attack vector to use other popular alternatives for the naming of the “response” object such as res, res2, response1, response 1, etc. If at some point the server responds with the expected data, then NodeXP correctly guessed the name of the object and it can proceed with exploitation.

Before analysing blind-based detection, firstly it is described how NodeXP evaluates the response and decides whether the application is vulnerable or not. As it was mentioned before, an attack vector is JavaScript code that executes a predetermined operation which can be one of the following: i) echoing a random number, ii) performing a mathematical operation and iii) string concatenation. Examples of such attack vectors are: i) eval(1201887702257507), ii) eval(12592*123) and iii) response.end(“12592”+“123”), respectively. The rationale is that if the application is indeed vulnerable to SSJI, then the result of the operation must be included in the response. For instance, if NodeXP generates the random number 1201887702257507, then it will create the attack vector eval(1201887702257507), which will be injected in the parameter of a vulnerable application. If this number is echoed in the response, then NodeXP assumes that the application is vulnerable to SSJI. The same reasoning applies to the other attack vector types. For instance in the case of a mathematical operation such as eval(12592*123), the response should include the result of the multiplication (i.e., 1593096).

Regardless of the actual operation of the attack vector, the expected response must be random enough (e.g., a non-sense alphanumeric string or the result of a calculation with many digits) in the sense that it must be very unlikely to be observed legitimately in an application. Otherwise, NodeXP may draw false-positive conclusions. For example, consider if an attack vector included the concatenation of the strings “log” and ”out”. Many application’s responses could contain the “logout” string legitimately. For this reason, NodeXP concatenates only random strings to avoid such erroneous decisions.

Blind-based detection is a more sophisticated technique. The main difference between result and blind-based SSJI lies in the way that the data is retrieved after the execution of the injected code. More specifically, as it was previously mentioned, the web application response may not be conclusive for NodeXP, because it may contain an error or it does not include the expected output (i.e., the result of the mathematical operation). In these cases, NodeXP tries to indirectly infer whether the injected code was executed by deliberately introducing time delays in the response. To achieve this, NodeXP uses attack vectors that halt the execution of the application (e.g., using the sleep() function or the Date object - See Figure 5-3). In this way, the response of the application is delayed based on a specified time duration. By measuring the time, it took the application to respond, NodeXP is able to identify if the code executed successfully.



```
var cur_date; var d = new Date(); do{ cur_date =
new Date();} while(cur_date-d <= #time#)
```

Figure 5-3 An attack vector for blind-based detection

In order to prove the validity of such delays, the calculation of a time threshold, based on the completion of valid HTTP requests, is considered necessary. To this end, NodeXP first makes multiple requests and computes the average response time of the web application. The average response time is considered as a reference point to decide whether the Node.js application is vulnerable or not. That is, NodeXP specifies a time delay greater than the average response time to ensure that delays in the response time are triggered by the injected attack vector and not due to network conditions (i.e., Jitter). Without the average response time, NodeXP would be subject to false positives, because it may erroneously identify a web application vulnerable to blind SSJI, but in reality the delay was caused due to network jitter and not due to execution of the attack vector payload.

An important notice here is that a major difference between blind SQL injections and blind-based SSJI is that the former (i.e., SQL injection) is utilized not only for vulnerability detection, but also for exploitation purposes. More specifically, a blind SQL injection is utilized to extract the contents of the database typically byte-by-byte, evidently a time consuming procedure. On the other hand, blind SSJI is used only to infer whether a web application is vulnerable or not. The exploitation part of SSJI does not involve any blind-based injection technique, since the main goal is the establishment of a remote connection by injecting the related shell. Hence, the scope of blind SQL injections is broader compared to their SSJI counterparts.

5.3.5 Exploitation Module

The exploitation module is triggered when the vulnerability detection module determines that the application is vulnerable. The module automatically or manually is capable of executing a Node.js-based shellcode by exploiting an SSJI vulnerability to establish a connection with the remote machine hosting the vulnerable application. The result is an interactive shell allowing remote access to the file system of the (remote) machine. Even more importantly, the interactive shell can execute any command/program on the remote machine with the same privileges of the vulnerable web application.

This module has three different shell connection types that cover different exploitation scenarios. The first shell is a Node.js-based reverse TCP shell, in which the remote web application initiates the connection to the attacker's machine. NodeXP can automatically upgrade this shell to the well-known Meterpreter (i.e., a feature-rich shell of Metasploit framework). The second is a Node.js-based bind TCP shell, which is the opposite of the reverse shell. In the bind shell, the attacker's machine initiates the connection to the vulnerable web application. The bind shell is not automatically upgraded to Meterpreter, but the user can do this manually. The third shell is a reverse SSL shell. Compared to its TCP counterpart, the reverse SSL shell establishes a secure communication between the attacker and the web application machine, bypassing in this way intrusion detection systems and firewalls that can block the connection. It should be noted that the reverse SSL shell works only with the Metasploit framework, while the previous two shells (reverse and bind TCP) can be utilized with or without Metasploit.

NodeXP can exploit the SSJI vulnerability not only to obtain a shell but perform enumeration activities as well. In particular, NodeXP can take advantage of Node.js functionalities to perform directory listing as well as access and read sensitive file contents (e.g., the /etc/passwd file) given sufficient permissions. Moreover, NodeXP can upload a file and even execute it. This capability aims to demonstrate that the impact of SSJIs can be disastrous as an attacker can upload executable files such as a bitcoin miner or ransomware to cause havoc without the need of a remote shell connection.

5.4 Evaluation

5.4.1 Evaluation Approach

To evaluate the detection and exploitation capabilities of NodeXP, first the attack vectors were created that will be used for the detection of SSJI vulnerabilities. All attack vectors are then processed one-by-one by the obfuscation engine to create the final form of the attack vectors that were used to detect SSJI vulnerabilities.

Afterwards, three comprehensive and diverse assessments were performed. In the first assessment, the detection capabilities of NodeXP were evaluated using a custom testbed that was developed for this purpose. The testbed contains a set of vulnerable Node.js applications. The vulnerabilities are inspired by real ones that have been discovered in real-world systems. In the second assessment, NodeXP was compared with other vulnerability scanners that support SSJI vulnerabilities against virtual lab applications. Finally, in the third assessment, NodeXP was evaluated against real-world applications to detect 0-day SSJI vulnerabilities in Node.js applications (i.e., vulnerabilities that have not been reported before).

5.4.2 First Assessment: Testbed Applications

The first assessment involves a custom testbed, which includes 9 Node.js applications based on different SSJI vulnerability scenarios. These applications can be considered as a baseline benchmark for the detection capabilities of NodeXP. The vulnerable applications are:

1. regular-get.js prints a message concatenated with the username. The username is transferred to the application via the GET “name” parameter. The vulnerability is due to the use of eval() function without input validation.
2. regular-post.js is similar to the regular-get.js, but instead of a GET parameter, it utilizes the “name” parameter as POST.
3. regular-blind.js is similar to the previous application, but in this case, the application does not send back a message, so there is no visual feedback.
4. regular-base64.js is based on the regular-post.js application, but it performs a base64 encoding on the user input.
5. whitespace.js application is similar to regular-post.js and strips from the “name” POST parameter any whitespace character (e.g., space, tab).
6. escape-chars.js application is like regular-post.js and strips from the “name” POST parameter the characters “&”, “\\”, “;”, “$”.
7. json-parse.js expects to receive a JSON object in the “name” POST parameter, which is parsed using eval() and its values are included in the response body.
8. function-constructor.js passes the value of the “name” POST parameter to the vulnerable Function() constructor. This is the only application that does not use the eval() function.
9. string-manipulation.js application is based on the vulnerability found in PayPal’s demo application. The basic flaw is that input validation is performed only when the input is a string. The input is passed to the application via the “name” GET parameter.

Using non-obfuscated attack vectors, NodeXP managed to detect and exploit the vulnerabilities of *regular-get.js, regular-post.js, regular-blind.js, json-parse.js, regular-base64.js,* and *function-constructor.js* without the need of obfuscation. On the other hand, the applications escape-chars.js and whitespace.js filter the user input to remove specific characters. In this case, only obfuscated attack vectors were able to bypass the filters of these applications. Finally, in string-manipulation.js the exploitation is based on utilizing the “name[ ]” GET parameter (so that the parameter is considered an array and not a string). This case required manual configuration in the NodeXP interface.

5.4.3 Second Assessment: Comparison with Vulnerability Scanners

In the second part of our evaluation, NodeXP was compared against state-of-the-art vulnerability scanners for web applications. Some of them are open-source and come at no cost such as ZAP [120], Vega [121], W3af [122], while others are commercial such as Acunetix [49], BurpSuite [50]. These tools are briefly presented in the following:

- Acunetix [49] is a popular vulnerability scanner for websites and web APIs. It detects more than 4500 web application vulnerabilities automatically.
- Burp Suite [50] is a widely used web application security testing software. Among other functionalities, Burp Suite contains a web application vulnerability scanner, which covers a plethora of vulnerabilities (e.g. XSS, SQLi, SSJI, etc.) and a JavaScript analysis engine that implements both static and dynamic techniques.
- ZAP (Zed Attack Proxy) [120] is a popular vulnerability scanner for web applications, which has been developed by the Open Web Application Security Project (OWASP). It offers automatic and manual security testing services to satisfy both inexperienced users and experienced pentesters.
- Vega [121] is a web security scanner and testing platform. It includes automated, manual, and hybrid security testing services.
- W3af (Web Application Attack and Audit Framework) [122] has been developed to help individuals secure their web applications. It provides an interface to discover and exploit more than 200 web application vulnerabilities.

To compare NodeXP against the previously mentioned vulnerability scanners, a set of free, open-source, web applications was collected that are vulnerable to SSJI. These vulnerable web applications are also called virtual-lab applications, because their main goal is to provide a safe and legal environment for developers to understand and learn web application security, as well as facilitate security professionals to test the effectiveness of their tools. The virtual-labs are discussed below:

- Nodegoat [123] is a vulnerable Node.js web application to demonstrate how the OWASP top 10 security risks[33] apply to Node.js web applications. Nodegoat is vulnerable to various attacks (e.g., XSS), but this thesis focuses only on the SSJI vulnerability, which is located in 3 parameters ("preTax", "afterTax", "roth") in the ”/contributions” page.
- Express TestBench [124] is also a Node.js application with deliberate vulnerabilities. The SSJI vulnerability is in the "name" POST parameter located in the "/serialization/node-serialize" page.
- XVNA (Extreme Vulnerable Node Application) [125] is a Node.js application that contains 8 different categories of vulnerabilities. This thesis focuses only on the SSJI vulnerability, which is located in the "/eval" page and more specifically in the "id2" GET parameter.
- node.nV [126] is a Node.js application that contains several web application vulnerabilities (e.g. XSS, SSJI, etc.). The SSJI vulnerability is located in the "/tools" page in the "code" GET parameter.
- Appsecco [127] is another vulnerable Node.js application.

The results of the comparison between NodeXP and its peers are presented in Table 20. It was observed that only NodeXP and Acunetix detected all the SSJI vulnerabilities in the applications. Burp Suite did not detect the (rather trivial) vulnerability of the Express TestBench application. The remaining vulnerability scanners Vega, W3af, and ZAP did not detect any of the vulnerabilities. In summary, NodeXP managed to overcome all the open-source vulnerability scanners and achieved the same results with Acunetix. Regarding the exploitation process, none of its peers was able to exploit the SSJI vulnerabilities; It is fair to mention that the majority of these tools (i.e., Acunetix, Burp Suite,

[33] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Vega, and ZAP) perform only vulnerability scanning and do not support exploitation (except for W3af). Nevertheless, it can be extrapolated from the results that NodeXP is the only free all-around tool to detect as well as exploit SSJI in an efficient and automated manner.

Table 20 Comparison Results for vulnerability discovery and exploitation feature

		Acunetix	Burp Suite	Vega	W3af	ZAP	NodeXP
App	*Nodegoat*	✓	✓	✗	✗	✗	✓
	Node TestBench	✓	✗	✗	✗	✗	✓
	XVNA	✓	✓	✗	✗	✗	✓
	node.nV	✓	✓	✗	✗	✗	✓
	Appsecco (RCE)	✓	✓	✗	✗	✗	✓
Feature	*Exploitation*	✗	✗	✗	✗	✗	✓

5.4.4 Third Assessment: 0-day Vulnerabilities

Finally, the capabilities of NodeXP against real-world applications were evaluated to discover 0-day vulnerabilities. To this end, 50 Node.js applications were downloaded in total from GitHub. Moreover, 10 out of these 50 applications were using the eval() function or/and the Function constructor in their JavaScript code. At this point, it is important to mention that discovering all the vulnerable Node.js applications on GitHub were out of the scope of this thesis. Instead, the main objective was to show that even in a small set of applications, it is possible to discover SSJI vulnerabilities.

NodeXP was tested against the subset of the downloaded Node.js applications (10 in total). NodeXP was able to discover a 0-day vulnerability in 1 application. This application passes the user-supplied data to the eval function without any input handling. More specifically, the name of the application was SubleasingUIUC, which is a Node.js application that provides to the students at the University of Illinois Urbana-Champaign subleasing and renting information. The application is deployed online[34], fact that makes the vulnerability even more crucial. Therefore, the vulnerability was responsibly disclosed and reported to the developers via their GitHub page[35].

To elaborate more on the technical details, the vulnerability is located in the rent/backend/routes/api.js file that takes two GET parameters, sort and select. NodeXP detected that both of these parameters are vulnerable to SSJI. As indicated in the code snippet below, the user-supplied data that contained to the sort and select parameters are passed to the eval function without any input handling, leaving the application unprotected to arbitrary code execution.

```
Apartment.count(eval("("+req.query.where+")"))
    .sort(eval("("+req.query.sort+")"))
    .select(eval("("+req.query.select+")"))
```

5.5 Countermeasures against SSJI attacks

In this section, the thesis proposes countermeasures that developers should adopt to secure applications from SSJI attacks. In general, the most secure practice is to avoid passing user input in functions that dynamically execute JavaScript code in server-side applications. This requires that developers to be aware of all instances where the application dynamically executes code using eval() and/or Function(). As this can be challenging due to the code complexity or time limitations, in some cases (depending on the application logic), the aforementioned functions can be replaced with alternative secure APIs of Node.js. For instance, instead of using the eval() function to parse a JSON

[34] http://subleasingatuiuc.herokuapp.co

[35] https://github.com/fji4/rent/issues/16#issue-557582773

request, the developer can deploy the JSON.parse method. If for some reason, eval() should be used for parsing JSON objects, then the application must validate the input using jsonschema.

In case it is necessary to process the user input with eval() or Function(), the developer should perform a careful input validation of the untrusted user input. The input validation refers to the procedure of filtering or blocking dangerous characters from the input data and is performed using two different methodologies: (a) whitelisting and (b) blacklisting.

Blacklisting technique searches the user input for malicious patterns before allowing its execution. To protect against SSJI attacks, the black- listing technique can strip off specific characters from the user input that are considered “dangerous” (i.e., ampersand (&), semicolon (;), single quote (’), double quote (”), etc.). The developer should be cautious to blacklist all the known “dangerous” characters. The drawbacks of this technique are twofold. First, its success is based on known malicious injection patterns. If an attacker discovers new variations of the SSJI attack - not included in the blacklist - the attack will launch successfully. Secondly, in the case of obfuscated user input, this technique will not manage to strip the blacklisted characters and the attack will succeed.

Whitelisting technique checks whether the user input matches some predefined safe input patterns. In case it does not, the input is rejected. This technique is more secure than blacklisting as it solves the problem of new attack variations by automatically blocking any input that does not match a safe input. Whitelists are commonly implemented using regular expressions that imply the safe format of the user input. A disadvantage of this technique is that regular expressions can be complex to implement. Furthermore, the developers should be careful when they implement whitelists to avoid filtering and blocking legitimate inputs.

It is worth mentioning that similarly to other programming languages, Node.js has several packages to validate user input, e.g., validator [128]. The use of such packages can significantly reduce the developers’ effort to write down filters. Moreover, in tandem with input validation, specialized security modules of Node.js can be placed to bolster security. For example, the VM2 module, which is an open-source sandbox[36], runs untrusted code securely in a single process and performs custom security checks, in order to prevent escaping from the sandbox environment.

Finally, another defensive technology that can be utilized for blocking SSJI attacks is Intrusion Detection Systems (IDS). The detection capabilities of Snort (version 2.9) and Suricata (version 4.1.8) have been evaluated, which are both well-known signature-based open-source IDS against NodeXP and SSJI generally. Suricata and Snort were deployed with their pre-installed signatures, which are created by the security community and third parties. It was observed that neither Suricata nor Snort were able to detect any of the plain (i.e., no obfuscation) SSJI attacks. This result can be directly attributed to the sheer lack of SSJI signatures pointing to the fact that SSJI attacks have been neglected by the security community. This thesis aims to raise awareness on the SSJI threat as the impact of such attacks is significant and can disrupt the workflow of organizations. To this end, this thesis has prepared a set of open-source Snort and Suricata signatures (see Appendix) that hopefully will facilitate the security community and organizations to better detect and respond to these attacks.

5.6 Discussion

As SSJI vulnerabilities pose a significant threat to Node.js applications, it is of utmost importance to implement a methodology that can be easily deployed and detect such vulnerabilities timely. This work proposed a methodology and its software implementation named NodeXP for the detection and exploitation of SSJI vulnerabilities in Node.js applications. NodeXP is developed to cover as many SSJI vulnerability variations as possible. Taking into account the security mechanisms that might be implemented on the server-side, the attack vectors are encoded by an obfuscation engine. A thorough

[36] https://github.com/patriksimek/vm2

evaluation of NodeXP on three different scenarios was performed. At first, NodeXP was evaluated on a custom testbed that contains 9 variations of SSJI vulnerabilities. Second, NodeXP's detection capability was compared with other state-of-the-art vulnerability scanners using virtual lab applications. The results showed that NodeXP overall presents better detection and exploitation capabilities compared to its peers. Last but not least, using NodeXP, it was discovered a Node.js application that contains a 0-day SSJI vulnerability.

As a testing tool, this thesis envisions that NodeXP can facilitate developers, penetration testers, and red teams to discover bugs and vulnerabilities related to SSJI attacks and improve the security posture of Node.js applications. As a general countermeasure, developers should provide safer Node.js applications by exercising defence in depth practices, that is by combining whitelisting and blacklisting with security modules (e.g., a sandbox), instead of relying only on one security technique.

6. Research Contributions

The research that has been conducted and presented in this thesis mainly formed four contributions (RC1-RC4) that can be classified into three primary areas: i) defending AI systems against AML attacks, ii) detecting sophisticated cyberattacks, and iii) vulnerability detection and exploitation of SSJI attacks in Node.js applications.

The evolution of AI methods has led to a new attack paradigm, where adversaries seek to affect the functionality of AI systems in order to confuse their output, known as AML (Adversarial Machine Learning). Several works exist in the literature that focus on methods which aim to compromise ML/DL models introducing novel ways that either targeting the training data or the model (e.g., evasion and poisoning attacks). However, during this thesis a knowledge gap has been identified regarding the existing defense methods against such attacks. Thus, to bridge this gap, Chapter 2 surveyed existing defense methods and introduced a taxonomy aiming to drive more research towards robust AI systems as well as facilitate the research community to develop innovative detection methods for AML attacks. This forms the fourth research contribution of this thesis:

RC1 A survey of defense methods against AML attacks.

Phishing email attacks and EKs have been risen the last couple of years affecting the security of both organizations and individuals. State-of-the-art existing works in the detection of such sophisticated cyberattacks mostly employ ML and DL techniques due to their efficacy on processing large data volumes and complex features. During this thesis, several gaps have been identified in the literature, such as limited ML/DL methods (i.e., most works deploy the same ML algorithms), evaluation experiments (i.e., usually the experimental evaluation is not based on realistic scenarios), obsolete datasets (i.e., in the phishing email detection domain the evaluation is performed on old phishing emails), doubtful generalization on newer unseen data (i.e., the majority of existing works do not prove whether their results can also be achieved on future attacks). Motivated by the aforementioned limitations, this thesis presented two novel methodologies, which thoroughly discussed in Chapters 3 and 4. The first methodology investigates for the first time the application of ensemble learning as well as the concepts of hybrid features in the phishing email detection domain. Specifically, the performance of stacking and soft-voting ensemble learning methods have been studied along with content-based and text-based features extracted from the emails. The second methodology provides a holistic ML-driven approach for the detection of EKs via their network activity. This methodology introduces the concept of *potential EK session* to separate the EK infection from the benign traffic and novel EK features that have been extracted from the network traces of the EK infection. Both methodologies have been evaluated on real-life assessments that take into account innovative evaluation rules to guarantee that a) the evaluation results are not biased and b) the evaluation limitations of existing works have been considered. The evaluation results (described in Chapter 3, section 3.4 and Chapter 4, section 4.6) concluded that both methodologies advanced the detection performance on the phishing email detection and EK detection domains. This forms the next two research contributions of this thesis:

RC2 An ensemble learning methodology for phishing email detection that exploits hybrid phishing email features.

RC3 A holistic ML-driven methodology for the detection of EKs via their network activity.

Node.js applications have gained a lot of popularity the recent years mainly due to their main characteristic, which is a unified development stack, which allows software engineers to work both at the user interface side of an application, as well as at the server-side using the same programming language, JavaScript. Due to the upward trend of Node.js, the shift of attackers toward finding Node.js vulnerabilities were quite natural and well expected. This quest has been proven very successful using code injection techniques. In this regard, this thesis proposed a novel platform for automatically detecting and exploiting SSJI attacks in Node.js applications, named NodeXP. The novel platform presented in Chapter 5 offered a solution to security professionals and developers to identify security gaps in Node.js applications. NodeXP provides novel aspects such as attack vector obfuscation that extend the robustness of previous vulnerability detection and exploitation scanners. The efficacy of NodeXP was evaluated against several scenarios and compared with previous scanners as discussed in Chapter 5 (section 5.4). This leads to the first main contribution of this thesis, which advances the existing knowledge regarding SSJI attacks as well as their detection and exploitation in a simple yet effective way:

RC4 Design and implementation of a novel platform for performing SSJI attacks in Node.js applications.

To summarize, RC1-4 provide a holistic view on closely related yet different fields varying from investigating the existing knowledge regarding the protection of AI systems (RC1), developing AI-driven methodologies for detecting sophisticated cyberattacks (RC2, RC3) as well as formalizing SSJI attacks to identify vulnerabilities in Node.js applications (RC4). Overall, all research contributions delve into the concept of “AI for Cybersecurity” and “Cybersecurity for AI”.

7. Conclusion and Future Work

This thesis was motivated by the fact that nowadays cyberattacks have been progressed to adopt novel methods and features that allow them to evade current detection solutions. For instance, phishing email attacks are known for more than a decade; however, the last couple of years they have been enhanced using AI-based text generation methods to generate emails' texts that are identical to benign. Moreover, even though AI methods like ML and DL have accomplished prominent results in cybersecurity and more specifically in the cyberattack detection domain, there are several gaps and limitations in the literature that affect the applicability of AI-based defense solutions in real-life conditions, which have been mentioned in existing works [15] [161]. These pitfalls are mostly related to the evaluation of ML and DL classifiers in the cyberattack detection domain (e.g., limited or outdated data employed in the assessment, experiments that do not represent real-life use cases, etc.) and have been considered in this thesis by introducing rules to avoid misleading results and performing real-life assessments (Chapter 3, section 3.4.1 and Chapter 4, section 4.6.1).

Furthermore, the heavy adoption of AI systems has led cybercriminals to introduce a new threat landscape, where they try to compromise ML and DL models to affect their output and reduce their efficacy (i.e., AML attacks). The literature includes numerous methods that focus on attacking either ML or DL models on various attack settings (e.g., white-box, black-box); however, limited research has been performed regarding enhancing the robustness of AI systems or detecting AML attacks. Thus, this thesis delves into the paradigm "AI for Cybersecurity" and "Cybersecurity for AI", namely AI-driven methodologies were proposed to tackle prominent cyberattacks (e.g., phishing and EKs) and cybersecurity methods have been studied to mitigate the ongoing threat of AML attacks and protect AI systems.

Summarizing, this thesis examines four research questions, which were answered by making four research contributions:

RQ1   *Is it possible to create a taxonomy of existing defenses against AML attacks?*

RC1   As it was presented in Chapter 2, it is possible to create a taxonomy based on the existing defenses against AML attacks. The taxonomy contains two main axis: a) mitigation and b) detection of AML attacks. The first axe focuses on methods, whose main objective is to enhance the robustness of ML and DL models (e.g., adversarial training) to mitigate AML attacks, while the second axe, collects solutions that deal with the detection of AML attacks. It is important to note that this is the first taxonomy in the AML field that focus specifically on the defense solutions and systematizes the existing knowledge on four different application domains, namely computer vision, cybersecurity, NLP, and audio.

RQ2   *Can the combination of Ensemble Learning with hybrid features, overall, enhance the phishing email detection performance?*

RC2   This thesis proved in Chapter 3 that Ensemble Learning, when deploying two methods (soft-voting and stacking) in combination with hybrid features, namely content-based and text-based features, can overall improve the phishing email detection performance. More specifically, the proposed Ensemble Learning methodology outperformed existing works and accomplished high detection performance (0.9942 F1-score, 0.9943 accuracy, precision, and recall, 0.967 MCC) on the largest imbalanced dataset (32,051 benign and 3,460 phishing emails) that has been used so far.

RQ3   *Can we build accurate EK detection systems that are capable of efficiently detecting state-of-the-art EKs based on traces at the network level operation of EKs?*

RC3   It has been shown in Chapter 4 that indeed it is possible to build robust EK detection systems exploiting the network traces that EKs leave behind. This was achieved by identifying features in the network traffic that when parsed to an ML algorithm it can effectively detect the EK infection procedure. To this end, 47 features were identified that deployed from EKnad, an innovative methodology that unfolds in two phases (pre-processing and detection) and based on the experimental evaluation is able to detect both existing and future EKs (the ML models tested on EK families that have not been used in the training) with high detection performance (at least 0.983 accuracy, 0.065 FPR, 0.984 recall & precision, 0.983 F1-score, 0.949 MCC, and 0.996 AUC) as well as to outperform well-known IDS (e.g., Snort and Suricata).

RQ4 Is it feasible to create a methodology to automatically detect and exploit SSJI vulnerabilities in Node.js applications?

RC4 This thesis concluded that it feasible to create a methodology that can automatically detect and exploit SSJI vulnerabilities in Node.js applications. Particularly, the thorough analysis of the SSJI attacks and the comprehension of their function led to the creation of a methodology, named NodeXP that is able to automatically detect and exploit SSJI vulnerabilities on Node.js applications by searching for unprotected variables in the applications and injecting arbitrary inputs to them. NodeXP were tested and compared with various similar applications (Acunetix, W3af, ZAP, etc.) as well as discovered a 0-day vulnerability on an open-source application (Chapter 5, section 5.4.4).

The contributions of this research can form the foundation for future research in the AI and cybersecurity domains with application in different fields (e.g., computer vision and NLP). Even though there exist several defense solutions in the literature designed to tackle AML attacks, the majority of these works are designed to work only on a particular application domain (e.g., audio) and provide defense only on a specific AML attack (e.g., JSMA). Currently, the research towards developing domain and attack agnostic defense solutions is at its infancy and the taxonomy introduced in Chapter 2 could facilitate the research community drive more research on this direction. Furthermore, given the findings of Chapter 3, Ensemble Learning resulted in improved detection performance in phishing email detection domain, which could be considered an encouraging step towards building accurate cyberattack detection solutions also for other domains, including but not limited to the detection of APTs, trojans, and ransomware.

Considering the findings of Chapters 3 and 4 regarding the evaluation of AI-driven solutions, where several pitfalls were identified, the creation of a holistic benchmark evaluation framework which will also take into account the models' robustness against AML attacks is of outmost importance. Establishing a holistic evaluation framework will facilitate researchers thoroughly assess their solutions under the same settings reducing common evaluation problems, such as inappropriate evaluation metrics and limited information about datasets and it will lead in the development of innovative AI-based solutions that will not stay on lab prototypes, but they will also be applied in real-life problems.

8. References

[1] C. Martignani, “Cybersecurity in cardiac implantable electronic devices,” *Expert Review of Medical Devices,* pp. 437-444, 2019.

[2] *Node.js code injection (RCE),* Accessed: October 2019.

[3] C.-A. Staicu, M. Pradel and B. Livshits, “Understanding and automatically preventing injection attacks on node.js,” 2016.

[4] Y. Wang and Z. Li, “SQL Injection Detection via Program Tracing and Machine Learning,” in *Internet and Distributed Computing Systems,* Berlin, 2012.

[5] S. Son, K. McKinley and V. Shmatikov, “Diglossia: Detecting code injection attacks with precision and efficiency,” 2013.

[6] A. Stasinopoulos, C. Ntantogian and C. Xenakis, “Commix: automating evaluation and exploitation of command injection vulnerabilities in Web applications,” *International Journal of Information Security,* vol. 18, p. 49–72, 01 February 2019.

[7] *Phishing Statistics Report 2021,* September 2021.

[8] *Securing the Enterprise in the COVID world, The State of Email Security,* April 2021.

[9] *2019 Phishing Statistics and Email Fraud Statistics,* Accessed: December 2021.

[10] *Enisa Threat Landscape 2020 - Phishing,* October 2020.

[11] *Interpol COVID-19 Cybercrime Analysis Report,* Accessed: September 2020.

[12] M. M. Yamin, M. Ullah, H. Ullah and B. Katt, “Weaponized AI for cyber attacks,” *Journal of Information Security and Applications,* vol. 57, p. 102722, 2021.

[13] Y. Li, Z. Yang, X. Chen, H. Yuan and W. Liu, “A stacking model using URL and HTML features for phishing webpage detection,” *Future Generation Computer Systems,* vol. 94, p. 27–39, 2019.

[14] C. M. R. Haider, A. Iqbal, A. H. Rahman and M. S. Rahman, “An ensemble learning based approach for impression fraud detection in mobile advertising,” *Journal of Network and Computer Applications,* vol. 112, p. 126–141, 2018.

[15] A. Das, S. Baki, A. E. Aassal, R. Verma and A. Dunbar, “SoK: A Comprehensive Reexamination of Phishing Research From the Security Perspective,” *IEEE Communications Surveys Tutorials,* vol. 22, pp. 671-708, 2020.

[16] A. El Aassal, S. Baki, A. Das and R. M. Verma, “An in-depth benchmarking and evaluation of phishing detection research for security needs,” *IEEE Access,* vol. 8, p. 22170–22192, 2020.

[17] T. Gangavarapu, C. D. Jaidhar and B. Chanduka, “Applicability of machine learning in spam and phishing email filtering: review and approaches.,” *Artificial Intelligence Review,* vol. 53, 2020.

[18] E. Suren and P. Angin, “Know Your EK: A Content and Workflow Analysis Approach for Exploit Kits,” *Journal of Internet Services and Information Security (JISIS),* vol. 9, p. 24–47, February 2019.

[19] D. Carlin, J. Burgess, P. O'Kane and S. Sezer, “You Could Be Mine(d): The Rise of Cryptojacking,” *IEEE Security & Privacy,* vol. 18, pp. 16-22, March 2020.

[20] D. Pliatsios, P. Sarigiannidis, K. Psannis, S. K. Goudos, V. Vitsas and I. Moscholios, “Big Data against Security Threats: The SPEAR Intrusion Detection System,” in *2020 3rd World Symposium on Communication Engineering (WSCE),* 2020.

[21] S. Yoo, S. Kim and B. B. Kang, “The Image Game: Exploit Kit Detection Based on Recursive Convolutional Neural Networks,” *IEEE Access,* vol. 8, pp. 18808-18821, 2020.

[22] M. Aldwairi, M. Hasan and Z. Balbahaith, “Detection of drive-by download attacks using machine learning approach,” in *Cognitive Analytics: Concepts, Methodologies, Tools, and Applications*, IGI Global, 2020, p. 1598–1611.

[23] E. Süren, “ZEKI: unsupervised zero-day exploit kit intelligence,” *Turkish Journal of Electrical Engineering & Computer Sciences,* vol. 28, p. 1859–1870, 2020.

[24] N. Akhtar and A. Mian, “Threat of adversarial attacks on deep learning in computer vision: A survey,” *Ieee Access,* vol. 6, p. 14410–14430, 2018.

[25] E. Anthi, L. Williams, A. Javed and P. Burnap, “Hardening machine learning denial of service (DoS) defences against adversarial attacks in IoT smart home networks,” *computers & security,* vol. 108, p. 102352, 2021.

[26] S. Kaviani, K. J. Han and I. Sohn, “Adversarial attacks and defenses on AI in medical imaging informatics: A survey,” *Expert Systems with Applications,* p. 116815, 2022.

[27] X. Wang, J. Li, X. Kuang, Y.-a. Tan and J. Li, “The security of machine learning in an adversarial setting: A survey,” *Journal of Parallel and Distributed Computing,* vol. 130, p. 12–23, 2019.

[28] S. Qiu, Q. Liu, S. Zhou and C. Wu, “Review of artificial intelligence adversarial attack and defense technologies,” *Applied Sciences,* vol. 9, p. 909, 2019.

[29] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis and G. Loukas, “A taxonomy and survey of attacks against machine learning,” *Computer Science Review,* vol. 34, p. 100199, 2019.

[30] C. Ntantogian, P. Bountakas, D. Antonaropoulos, C. Patsakis and C. Xenakis, “NodeXP: NOde. js server-side JavaScript injection vulnerability DEtection and eXPloitation,” *Journal of Information Security and Applications,* vol. 58, p. 102752, 2021.

[31] P. Bountakas and C. Xenakis, “HELPHED: Hybrid Ensemble Learning PHishing Email Detection,” *Journal of Network and Computer Applications,* vol. 210, p. 103545, 2023.

[32] P. Bountakas, C. Ntantogian and C. Xenakis, “EKnad: Exploit Kits’ network activity detection,” *Future Generation Computer Systems,* vol. 134, p. 219–235, 2022.

[33] P. Bountakas, K. Koutroumpouchos and C. Xenakis, “A Comparison of Natural Language

Processing and Machine Learning Methods for Phishing Email Detection," in *The 16th International Conference on Availability, Reliability and Security*, 2021.

[34] A. Ojamaa and K. Düüna, "Security Assessment of Node.js Platform," in *Information Systems Security*, Berlin, 2012.

[35] D. Antonaropoulos, *NodeXP - An automated and integrated tool for detecting and exploiting Server Side JavaScript Injection vulnerability on Node.js services*, 2018.

[36] B. Sullivan, *Server-Side JavaScript Injection*, BlackHat, 2011.

[37] J. Davis, G. Kildow and D. Lee, "The Case of the Poisoned Event Handler: Weaknesses in the Node.Js Event-Driven Architecture," in *Proceedings of the 10th European Workshop on Systems Security*, New York, NY, USA, 2017.

[38] C.-A. Staicu and M. Pradel, "Freezing the Web: A Study of ReDoS Vulnerabilities in Javascript-based Web Servers," in *Proceedings of the 27th USENIX Conference on Security Symposium*, Berkeley, 2018.

[39] C.-A. Staicu, M. Pradel and B. Livshits, "SYNODE: Understanding and Automatically Preventing Injection Attacks on NODE.JS," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018*, San Diego, California, USA, 2018.

[40] F. Gauthier, B. Hassanshahi and A. Jordan, "AFFOGATO: Runtime Detection of Injection Attacks for Node.Js," in *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*, New York, NY, USA, 2018.

[41] G. Richards, C. Hammer, B. Burg and J. Vitek, "The eval that men do," in *European Conference on Object-Oriented Programming*, Berlin, 2011.

[42] R. Sekar, "An Efficient Black-box Technique for Defeating Web Application Attacks," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*, 2009.

[43] H.-Y. Shih, H.-L. Lu, C.-C. Yeh, H.-C. Hsiao and S.-K. Huang, "A Generic Web Application Testing and Attack Data Generation Method," in *International Conference on Security with Intelligent Computing and Big-data Services*, Cham, 2017.

[44] S. Gupta and B. B. Gupta, "JS-SAN: defense mechanism for HTML5-based web applications against javascript code injection vulnerabilities," *Security and Communication Networks,* vol. 9, 2016.

[45] S. Gupta, B. B. Gupta and P. Chaudhary, "Enhancing the Browser-Side Context-Aware Sanitization of Suspicious HTML5 Code for Halting the DOM-Based XSS Vulnerabilities in Cloud," *International Journal of Cloud Applications and Computing (IJCAC),* vol. 7, 2017.

[46] R. Yan, X. Xiao, G. Hu, S. Peng and Y. Jiang, "New deep learning method to detect code injection attacks on hybrid applications," *Journal of Systems and Software,* vol. 137, pp. 67-77, 2018.

[47] V. Prokhorenko, K.-K. R. Choo and H. Ashman, "Web application protection techniques: A taxonomy," *Journal of Network and Computer Applications,* vol. 60, p. 95–112, 2016.

[48] S. Gupta and B. B. Gupta, “Detection, Avoidance, and Attack Pattern Mechanisms in Modern Web Application Vulnerabilities: Present and Future Challenges,” *Int. J. Cloud Appl. Comput.,* vol. 7, p. 1–43, July 2017.

[49] *Acunetix Vulnerability Scanner,* Accessed: October 2019.

[50] *Burp Suite Vulnerability Scanner,* Accessed: October 2019.

[51] P. Schläpfer, *From PoC to Exploit Kit: Purple Fox now exploits CVE-2021-26411,* 2021.

[52] J. Burgess, P. O’Kane, S. Sezer and D. Carlin, “LSTM RNN: detecting exploit kits using redirection chain sequences,” *Cybersecurity,* vol. 4, p. 1–15, 2021.

[53] Y. Qin, W. Wang, S. Zhang and K. Chen, “An Exploit Kits Detection Approach Based on HTTP Message Graph,” *IEEE Transactions on Information Forensics and Security,* vol. 16, p. 3387–3400, 2021.

[54] N. Provos, P. Mavrommatis, M. A. Rajab and F. Monrose, “All Your iFRAMEs Point to Us,” in *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, 2008.

[55] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage and G. M. Voelker, “Manufacturing Compromise: The Emergence of Exploit-as-a-service,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, New York, NY, USA, 2012.

[56] M. Musch and M. Johns, “U Can't Debug This: Detecting {JavaScript} {Anti-Debugging} Techniques in the Wild,” in *30th USENIX Security Symposium (USENIX Security 21),* 2021.

[57] L. Ma, B. Ofoghi, P. Watters and S. Brown, “Detecting Phishing Emails Using Hybrid Features,” in *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, 2009.

[58] U. Bhowan, M. Johnston, M. Zhang and X. Yao, “Evolving Diverse Ensembles Using Genetic Programming for Classification With Unbalanced Data,” *IEEE Transactions on Evolutionary Computation,* vol. 17, pp. 368-386, 2013.

[59] I. R. A. Hamid and J. Abawajy, “Hybrid feature selection for phishing email detection,” in *International Conference on Algorithms and Architectures for Parallel Processing,* 2011.

[60] *Jose Nazario Phishing Email Corpus,* Accessed: November 2020.

[61] N. Moradpoor, B. Clavie and B. Buchanan, “Employing machine learning techniques for detection and classification of phishing emails,” in *2017 Computing Conference*, 2017.

[62] *Enron Email Dataset,* Accessed: November 2020.

[63] A. Akinyelu and A. Adewumi, “Classification of Phishing Email Using Random Forest Machine Learning Technique,” *Journal of Applied Mathematics,* vol. 2014, April 2014.

[64] S. Smadi, N. Aslam, L. Zhang, R. Alasem and M. A. Hossain, “Detection of phishing emails using data mining algorithms,” in *2015 9th International Conference on Software, Knowledge,*

Information Management and Applications (SKIMA), 2015.

[65] S. Marchal, J. François, R. State and T. Engel, “PhishStorm: Detecting Phishing With Streaming Analytics,” *IEEE Transactions on Network and Service Management,* vol. 11, pp. 458-471, 2014.

[66] I. Fette, N. Sadeh and A. Tomasic, “Learning to detect phishing emails,” in *Proceedings of the 16th international conference on World Wide Web,* 2007.

[67] M. Chandrasekaran, K. Narayanan and S. Upadhyaya, “Phishing email detection based on structural properties,” in *NYS cyber security conference,* 2006.

[68] S. Abu-Nimeh, D. Nappa, X. Wang and S. Nair, “A comparison of machine learning techniques for phishing detection,” in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit,* 2007.

[69] A. Alhogail and A. Alsabih, “Applying machine learning and natural language processing to detect phishing email,” *Computers & Security,* vol. 110, p. 102414, 2021.

[70] D. Radev, “Clair collection of fraud email, acl data and code repository,” *ADCR2008T001,* 2008.

[71] E. S. Gualberto, R. T. De Sousa, P. D. B. Thiago, J. P. C. L. Da Costa and C. G. Duque, “From feature engineering and topics models to enhanced prediction rates in phishing detection,” *Ieee Access,* vol. 8, p. 76368–76385, 2020.

[72] E. S. Gualberto, R. T. De Sousa, T. P. D. B. Vieira, J. P. C. L. Da Costa and C. G. Duque, “The Answer is in the Text: Multi-Stage Methods for Phishing Detection Based on Feature Engineering,” *IEEE Access,* vol. 8, p. 223529–223547, 2020.

[73] *Spam Assassin Project (2015) Spam Assassin Public Corpus,* Accessed: November 2020.

[74] Y. Fang, C. Zhang, C. Huang, L. Liu and Y. Yang, “Phishing Email Detection Using Improved RCNN Model With Multilevel Vectors and Attention Mechanism,” *IEEE Access,* vol. 7, pp. 56329-56340, 2019.

[75] T. Mikolov, K. Chen, G. Corrado and J. Dean, *Efficient Estimation of Word Representations in Vector Space,* 2013.

[76] M. Hiransha, N. Unnithan, R. Vinayakumar and S. Kp, “Deep Learning Based Phishing E-mail Detection CEN-Deepspam,” 2018.

[77] C. N. Gutierrez, T. Kim, R. D. Corte, J. Avery, D. Goldwasser, M. Cinque and S. Bagchi, “Learning from the Ones that Got Away: Detecting New Forms of Phishing Attacks,” *IEEE Transactions on Dependable and Secure Computing,* vol. 15, pp. 988-1001, 2018.

[78] G. Egozi and R. Verma, “Phishing Email Detection Using Robust NLP Techniques,” in *2018 IEEE International Conference on Data Mining Workshops (ICDMW),* 2018.

[79] L. Halgaš, I. Agrafiotis and J. Nurse, “Catching the Phish: Detecting Phishing Attacks Using Recurrent Neural Networks (RNNs),” 2020, pp. 219-233.

[80] R. Verma, N. Shashidhar and N. Hossain, “Detecting phishing emails the natural language

way," in *European Symposium on Research in Computer Security*, 2012.

[81] A. Bergholz, G. Paaß, F. Reichartz, S. Strobel and S. Birlinghoven, "Improved phishing detection using model-based features," in *In Fifth Conference on Email and Anti-Spam, CEAS,* 2008.

[82] N. A. Unnithan, N. B. Harikrishnan, S. Akarsh, R. Vinayakumar and K. P. Soman, "Machine learning based phishing e-mail detection," *Security-CEN@ Amrita,* p. 65–69, 2018.

[83] N. A. Unnithan, N. B. Harikrishnan, R. Vinayakumar, K. P. Soman and S. Sundarakrishna, "Detecting phishing E-mail using machine learning techniques," in *Proc. 1st Anti-Phishing Shared Task Pilot 4th ACM IWSPA Co-Located 8th ACM Conf. Data Appl. Secur. Privacy (CODASPY),* 2018.

[84] S. Y. Yerima and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," *IEEE transactions on cybernetics,* vol. 49, p. 453–466, 2018.

[85] H. Zhang, G. Liu, T. W. S. Chow and W. Liu, "Textual and visual content-based anti-phishing: a Bayesian approach," *IEEE transactions on neural networks,* vol. 22, p. 1532–1546, 2011.

[86] Z.-H. Zhou, "Ensemble learning," in *Machine learning,* Springer, 2021, p. 181–210.

[87] M. Al-Sarem, F. Saeed, Z. G. Al-Mekhlafi, B. A. Mohammed, T. Al-Hadhrami, M. T. Alshammari, A. Alreshidi and T. S. Alshammari, "An Optimized Stacking Ensemble Model for Phishing Websites Detection," *Electronics,* vol. 10, 2021.

[88] H. Mekky, R. Torres, Z. Zhang, S. Saha and A. Nucci, "Detecting malicious HTTP redirections using trees of user browsing activity," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014.

[89] T. Taylor, X. Hu, T. Wang, J. Jang, M. P. Stoecklin, F. Monrose and R. Sailer, "Detecting Malicious Exploit Kits Using Tree-based Similarity Searches," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, New York, NY, USA, 2016.

[90] I. Nikolaev, M. Grill and V. Valeros, "Exploit Kit Website Detection Using HTTP Proxy Logs," in *Proceedings of the Fifth International Conference on Network, Communication and Computing,* New York, NY, USA, 2016.

[91] M. Traffic-Analysis, *Malicious Network Traffic,* 2021.

[92] S. Harnmetta and S. Ngamsuriyaroj, "Classification of Exploit-Kit behaviors via machine learning approach," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, 2018.

[93] J. Burgess, D. Carlin, P. O'Kane and S. Sezer, "REdiREKT: Extracting Malicious Redirections from Exploit Kit Traffic," in *2020 IEEE Conference on Communications and Network Security (CNS),* 2020.

[94] B. Eshete, A. Alhuzali, M. Monshizadeh, P. A. Porras, V. N. Venkatakrishnan and V. Yegneswaran, "EKHunter: A Counter-Offensive Toolkit for Exploit Kit Infiltration.," in *NDSS,* 2015.

[95] G. De Maio, A. Kapravelos, Y. Shoshitaishvili, C. Kruegel and G. Vigna, "PExy: The Other

Side of Exploit Kits," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, Cham, 2014.

[96] C. Curtsinger, B. Livshits, B. Zorn and C. Seifert, "ZOZZLE: Fast and Precise In-browser JavaScript Malware Detection," in *Proceedings of the 20th USENIX Conference on Security*, Berkeley, 2011.

[97] K. Rieck, T. Krueger and A. Dewald, "Cujo: Efficient Detection and Prevention of Drive-by-download Attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, New York, NY, USA, 2010.

[98] D. Canali, M. Cova, G. Vigna and C. Kruegel, "Prophiler: A Fast Filter for the Large-scale Detection of Malicious Web Pages," in *Proceedings of the 20th International Conference on World Wide Web*, New York, NY, USA, 2011.

[99] S. Kim and B. B. Kang, "FriSM: Malicious Exploit Kit Detection via Feature-Based String-Similarity Matching," in *Security and Privacy in Communication Networks*, Cham, 2018.

[100] Y. Li, Y. Wang, Y. Wang, L. Ke and Y.-a. Tan, "A feature-vector generative adversarial network for evading PDF malware classifiers," *Information Sciences*, vol. 523, pp. 38-48, 2020.

[101] L. Zilong, S. Yong and Z. Xue, "Idsgan: Generative adversarial networks for attack generation against intrusion detection," in *In Advances in Knowledge Discovery and Data Mining: 26th Pacific-Asia Conference*, Chengdu, China, 2022.

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

[103] 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.

[104] B. Biggio, B. Nelson and P. Laskov, "Support vector machines under adversarial label noise," in *Asian conference on machine learning*, 2011.

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

[106] B. Flowers, R. M. Buehrer and W. C. Headley, "Evaluating adversarial evasion attacks in the context of wireless communications," *IEEE Transactions on Information Forensics and Security*, vol. 15, p. 1102–1113, 2019.

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

[108] R. Abou Khamis, M. O. Shafiq and A. Matrawy, "Investigating resistance of deep learning-based ids against adversaries using min-max optimization," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, 2020.

[109] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European symposium on security and privacy (EuroS&P)*, 2016.

[110] S.-M. Moosavi-Dezfooli, A. Fawzi and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[111] A. Kurakin, I. J. Goodfellow and S. Bengio, “Adversarial examples in the physical world,” in *Artificial intelligence safety and security,* Chapman and Hall/CRC, 2018, p. 99–112.

[112] P. Xu, B. Kolosnjaji, C. Eckert and A. Zarras, “MANIS: Evading Malware Detection System on Graph Structure,” in *Proceedings of the 35th Annual ACM Symposium on Applied Computing,* New York, NY, USA, 2020.

[113] R. Abou Khamis and A. Matrawy, “Evaluation of adversarial training on different types of neural networks in deep learning-based idss,” in *2020 international symposium on networks, computers and communications (ISNCC),* 2020.

[114] G. Apruzzese, M. Andreolini, M. Marchetti, A. Venturi and M. Colajanni, “Deep reinforcement adversarial learning against botnet evasion attacks,” *IEEE Transactions on Network and Service Management,* vol. 17, p. 1975–1987, 2020.

[115] E. Anthi, L. Williams, M. Rhode, P. Burnap and A. Wedgbury, “Adversarial attacks on machine learning cybersecurity defences in industrial control systems,” *Journal of Information Security and Applications,* vol. 58, p. 102717, 2021.

[116] I. Debicha, T. Debatty, J.-M. Dricot and W. Mees, “Adversarial training for deep learning-based intrusion detection systems,” *arXiv preprint arXiv:2104.09852,* 2021.

[117] M. M. Rashid, J. Kamruzzaman, M. M. Hassan, T. Imam, S. Wibowo, S. Gordon and G. Fortino, “Adversarial Training for Deep Learning-based Cyberattack Detection in IoT-based Smart City Applications,” *Computers & Security,* p. 102783, 2022.

[118] *NodeXP - Detection and Exploitation Tool for Node.js Services,* Accessed: October 2019.

[119] *Express API documentation,* Accessed: October 2020.

[120] *OWASP Zed Attack Proxy,* Accessed: October 2019.

[121] *Vega Vulnerability Scanner,* Accessed: October 2019.

[122] *Web Application Attack and Audit Framework,* Accessed: October 2019.

[123] *NodeGoat: Vulnerable Node.js Application,* Accessed: October 2019.

[124] *Express TestBench: Intentionally Vulnerable Node Applications,* Accessed: October 2019.

[125] *Extreme Vulnerable Node Application,* Accessed: October 2019.

[126] *Intentionally Vulnerable Node.js Application,* Accessed: October 2019.

[127] *Simple Node app with an RCE,* Accessed: October 2019.

[128] *Node Package Manager Validator,* Accessed: October 2019.

[129] M. Anandarajan, C. Hill and T. Nolan, “Text preprocessing,” in *Practical Text Analytics*, Springer, 2019, p. 45–59.

[130] I. Feinerer and K. Hornik, “wordnet: WordNet Interface,” 2020.

[131] J. Ramos and others, “Using tf-idf to determine word relevance in document queries,” in *Proceedings of the first instructional conference on machine learning*, 2003.

[132] I. Guyon and A. Elisseeff, “An introduction to variable and feature selection,” *Journal of machine learning research,* vol. 3, p. 1157–1182, 2003.

[133] B. Alotaibi and M. Alotaibi, “Consensus and majority vote feature selection methods and a detection technique for web phishing,” *Journal of Ambient Intelligence and Humanized Computing,* vol. 12, p. 717–727, 2021.

[134] T. G. Dietterich, “Ensemble methods in machine learning,” in *International workshop on multiple classifier systems*, 2000.

[135] A. Das, S. Baki, A. El Aassal, R. Verma and A. Dunbar, “SoK: a comprehensive reexamination of phishing research from the security perspective,” *IEEE Communications Surveys & Tutorials,* vol. 22, p. 671–708, 2019.

[136] Z. Dou, I. Khalil, A. Khreishah, A. Al-Fuqaha and M. Guizani, “Systematization of knowledge (sok): A systematic review of software-based web phishing detection,” *IEEE Communications Surveys & Tutorials,* vol. 19, p. 2797–2819, 2017.

[137] E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro and K. Rieck, “Dos and Don\textquoterightts of Machine Learning in Computer Security,” in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, 2022.

[138] R. M. Verma, V. Zeng and H. Faridi, “Data Quality for Security Challenges: Case Studies of Phishing, Malware and Intrusion Detection Datasets,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security,* New York, NY, USA, 2019.

[139] B. Klimt and Y. Yang, “The enron corpus: A new dataset for email classification research,” in *European Conference on Machine Learning*, 2004.

[140] G. Kessler, “Virtual business: An Enron email corpus study,” *Journal of Pragmatics,* vol. 42, p. 262–270, 2010.

[141] M. Bekkar, H. K. Djemaa and T. A. Alitouche, “Evaluation measures for models assessment over imbalanced data sets,” *J Inf Eng Appl,* vol. 3, 2013.

[142] A. Aljofey, Q. Jiang, A. Rasool, H. Chen, W. Liu, Q. Qu and Y. Wang, “An effective detection approach for phishing websites using URL and HTML features,” *Scientific Reports,* vol. 12, p. 1–19, 2022.

[143] P. Bountakas, *HELPHED's Data,* 2021.

[144] R. Islam and J. Abawajy, “A multi-tier phishing detection and filtering approach,” *Journal of Network and Computer Applications,* vol. 36, p. 324–335, 2013.

[145] D. P. Yadav, P. Paliwal, D. Kumar and R. Tripathi, “A novel ensemble based identification of phishing e-mails,” in *Proceedings of the 9th International Conference on Machine Learning and Computing,* 2017.

[146] P. Laperdrix, O. Starov, Q. Chen, A. Kapravelos and N. Nikiforakis, “Fingerprinting in style: Detecting browser extensions via injected style sheets,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[147] I. Siniosoglou, P. Radoglou-Grammatikis, G. Efstathopoulos, P. Fouliras and P. Sarigiannidis, “A unified deep learning anomaly detection and classification approach for smart grid environments,” *IEEE Transactions on Network and Service Management,* 2021.

[148] Malwarebytes, *State of Malware Report 2020,* 2020.

[149] Malwarebytes, *Exploit Kits Fall 2019 Review,* 2019.

[150] B. Eshete and V. N. Venkatakrishnan, “WebWinnow: Leveraging Exploit Kit Workflows to Detect Malicious Urls,” in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy,* New York, NY, USA, 2014.

[151] H. B. Kazemian and S. Ahmed, “Comparisons of machine learning techniques for detecting malicious webpages,” *Expert Systems with Applications,* vol. 42, p. 1166–1177, 2015.

[152] V. Kotov and F. Massacci, “Anatomy of Exploit Kits,” in *Engineering Secure Software and Systems,* Berlin, 2013.

[153] L. Xu, Z. Zhan, S. Xu and K. Ye, “Cross-layer Detection of Malicious Websites,” in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy,* New York, NY, USA, 2013.

[154] C. Larsen, *Top 20 Shady Top Level Domains,* 2018.

[155] D. A. A. Gnana, S. A. A. Balamurugan and E. J. Leavline, “Literature review on feature selection methods for high-dimensional data,” *International Journal of Computer Applications,* vol. 975, p. 8887, 2016.

[156] Kurniabudi, D. Stiawan, Darmawijoyo, M. Y. Bin Idris, A. M. Bamhdi and R. Budiarto, “CICIDS-2017 Dataset Feature Analysis With Information Gain for Anomaly Detection,” *IEEE Access,* vol. 8, pp. 132911-132921, 2020.

[157] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain and A. J. Aljaaf, “A systematic review on supervised and unsupervised machine learning algorithms for data science,” *Supervised and unsupervised learning for data science,* p. 3–21, 2020.

[158] Tin Kam Ho, “Random decision forests,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition,* 1995.

[159] N. Friedman, D. Geiger and M. Goldszmidt, “Bayesian Network Classifiers,” *Machine Learning,* vol. 29, p. 131–163, 01 November 1997.

[160] F. Casino, K.-K. R. Choo and C. Patsakis, “HEDGE: Efficient Traffic Classification of Encrypted and Compressed Packets,” *IEEE Transactions on Information Forensics and Security,* vol. 14, pp. 2916-2926, 2019.

[161] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder and L. Cavallaro, “TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time,” in *28th USENIX Security Symposium (USENIX Security 19),* Santa, 2019.

[162] M. Botacin, F. Ceschin, R. Sun, D. Oliveira and A. Grégio, “Challenges and pitfalls in malware research,” *Computers & Security,* vol. 106, p. 102287, 2021.

[163] P. Bountakas, *EKnad's Datasets,* 2021.

[164] R. Kohavi, “A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2,* San Francisco, CA, USA, 1995.

[165] S. Huang, J. Yang, S. Fong and Q. Zhao, “Artificial intelligence in cancer diagnosis and prognosis: Opportunities and challenges,” *Cancer letters,* vol. 471, p. 61–71, 2020.

[166] C. J. Hernández-Castro, Z. Liu, A. Serban, I. Tsingenopoulos and W. Joosen, “Adversarial machine learning,” in *Security and Artificial Intelligence*, Springer, 2022, p. 287–312.

[167] R. Huang, B. Xu, D. Schuurmans and C. Szepesvári, “Learning with a strong adversary,” *arXiv preprint arXiv:1511.03034,* 2015.

[168] A. Madry, A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *arXiv preprint arXiv:1706.06083,* 2017.

[169] S. Sankaranarayanan, A. Jain, R. Chellappa and S. N. Lim, “Regularizing deep networks using efficient layerwise adversarial training,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[170] A. Shafahi, M. Najibi, M. A. Ghiasi, Z. Xu, J. Dickerson, C. Studer, L. S. Davis, G. Taylor and T. Goldstein, “Adversarial training for free!,” *Advances in Neural Information Processing Systems,* vol. 32, 2019.

[171] C. Xie, Y. Wu, L. v. d. Maaten, A. L. Yuille and K. He, “Feature Denoising for Improving Adversarial Robustness,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* 2019.

[172] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh and P. McDaniel, “Ensemble adversarial training: Attacks and defenses,” *arXiv preprint arXiv:1705.07204,* 2017.

[173] S. Lal, S. U. Rehman, J. H. Shah, T. Meraj, H. T. Rauf, R. Damaševičius, M. A. Mohammed and K. H. Abdulkareem, “Adversarial attack and defence through adversarial training and feature fusion for diabetic retinopathy recognition,” *Sensors,* vol. 21, p. 3922, 2021.

[174] F. Ding, K. Yu, Z. Gu, X. Li and Y. Shi, “Perceptual enhancement for autonomous vehicles: restoring visually degraded images for context prediction via adversarial training,” *IEEE Transactions on Intelligent Transportation Systems,* 2021.

[175] H. Xu, X. Liu, Y. Li, A. Jain and J. Tang, “To be robust or to be fair: Towards fairness in adversarial training,” in *International Conference on Machine Learning,* 2021.

[176] X. Fu, N. Zhou, L. Jiao, H. Li and J. Zhang, “The robust deep learning–based schemes for intrusion detection in Internet of Things environments,” *Annals of Telecommunications,* vol. 76, p. 273–285, 2021.

[177] J. Vitorino, N. Oliveira and I. Praça, “Adaptative Perturbation Patterns: Realistic Adversarial Learning for Robust Intrusion Detection,” *Future Internet,* vol. 14, p. 108, 2022.

[178] T. Miyato, A. M. Dai and I. Goodfellow, “Adversarial training methods for semi-supervised text classification,” *arXiv preprint arXiv:1605.07725,* 2016.

[179] S. Kitada and H. Iyatomi, “Attention meets perturbations: Robust and interpretable attention with adversarial training,” *IEEE Access,* vol. 9, p. 92974–92985, 2021.

[180] G. Bekoulis, J. Deleu, T. Demeester and C. Develder, “Adversarial training for multi-context joint entity and relation extraction,” *arXiv preprint arXiv:1808.06876,* 2018.

[181] C. Zhu, Y. Cheng, Z. Gan, S. Sun, T. Goldstein and J. Liu, “Freelb: Enhanced adversarial training for natural language understanding,” *arXiv preprint arXiv:1909.11764,* 2019.

[182] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805,* 2018.

[183] X. Liu, H. Cheng, P. He, W. Chen, Y. Wang, H. Poon and J. Gao, “Adversarial training for large neural language models,” *arXiv preprint arXiv:2004.08994,* 2020.

[184] J. Y. Yoo and Y. Qi, “Towards improving adversarial training of nlp models,” *arXiv preprint arXiv:2109.00544,* 2021.

[185] L. Pan, C.-W. Hang, A. Sil and S. Potdar, “Improved text classification via contrastive adversarial training,” in *Proceedings of the AAAI Conference on Artificial Intelligence,* 2022.

[186] M. Moradi and M. Samwald, “Improving the robustness and accuracy of biomedical language models through adversarial training,” *Journal of Biomedical Informatics,* vol. 132, p. 104114, 2022.

[187] M. Pal, A. Jati, R. Peri, C.-C. Hsu, W. AbdAlmageed and S. Narayanan, “Adversarial defense for deep speaker recognition using hybrid adversarial training,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 2021.

[188] H. Zhang and J. Wang, “Defense against adversarial attacks using feature scattering-based adversarial training,” *Advances in Neural Information Processing Systems,* vol. 32, 2019.

[189] S. Joshi, S. Kataria, Y. Shao, P. Zelasko, J. Villalba, S. Khudanpur and N. Dehak, *Defense against Adversarial Attacks on Hybrid Speech Recognition using Joint Adversarial Fine-tuning with Denoiser,* arXiv, 2022.

[190] G. Hinton, O. Vinyals, J. Dean and others, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531,* vol. 2, 2015.

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

[192] M. Goldblum, L. Fowl, S. Feizi and T. Goldstein, “Adversarially robust distillation,” in *Proceedings of the AAAI Conference on Artificial Intelligence,* 2020.

[193] M. Soll, T. Hinz, S. Magg and S. Wermter, “Evaluating defensive distillation for defending text processing neural networks against adversarial examples,” in *International Conference on Artificial Neural Networks,* 2019.

[194] S. Samanta and S. Mehta, “Towards crafting text adversarial samples,” *arXiv preprint arXiv:1707.02812,* 2017.

[195] G. Apruzzese, M. Andreolini, M. Colajanni and M. Marchetti, “Hardening random forest cyber detectors against adversarial attacks,” *IEEE Transactions on Emerging Topics in Computational Intelligence,* vol. 4, p. 427–439, 2020.

[196] G. Apruzzese, M. Andreolini, M. Marchetti, V. G. Colacino and G. Russo, “AppCon: Mitigating evasion attacks to ML cyber detectors,” *Symmetry,* vol. 12, p. 653, 2020.

[197] H. Jiang, J. Lin and H. Kang, “FGMD: A robust detector against adversarial attacks in the IoT network,” *Future Generation Computer Systems,* vol. 132, p. 194–210, 2022.

[198] S. Joshi, J. Villalba, P. Żelasko, L. Moro-Velázquez and N. Dehak, “Study of pre-processing defenses against adversarial attacks on state-of-the-art speaker recognition systems,” *IEEE Transactions on Information Forensics and Security,* vol. 16, p. 4811–4826, 2021.

[199] P. Samangouei, M. Kabkab and R. Chellappa, “Defense-gan: Protecting classifiers against adversarial attacks using generative models,” *International Conference on Learning Representations,* 2018.

[200] I. Rosenberg, A. Shabtai, Y. Elovici and L. Rokach, “Defense methods against adversarial examples for recurrent neural networks,” *arXiv preprint arXiv:1901.09963,* 2019.

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

[202] 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,* vol. 16, p. 711–724, 2017.

[203] M. Esmaeilpour, P. Cardinal and A. L. Koerich, “A robust approach for securing audio classification against adversarial attacks,” *IEEE Transactions on information forensics and security,* vol. 15, p. 2147–2159, 2019.

[204] D. Han, Z. Wang, Y. Zhong, W. Chen, J. Yang, S. Lu, X. Shi and X. Yin, “Evaluating and improving adversarial robustness of machine learning-based network intrusion detectors,” *IEEE Journal on Selected Areas in Communications,* vol. 39, p. 2632–2647, 2021.

[205] S. Weerasinghe, T. Alpcan, S. M. Erfani and C. Leckie, “Defending support vector machines against data poisoning attacks,” *IEEE Transactions on Information Forensics and Security,* vol. 16, p. 2566–2578, 2021.

[206] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu and B. Li, “Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach,” *computers & security,* vol. 73, p. 326–344, 2018.

[207] M. Pawlicki, M. Choraś and R. Kozik, “Defending network intrusion detection systems against adversarial evasion attacks,” *Future Generation Computer Systems,* vol. 110, p. 148–154, 2020.

[208] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri and M. Conti, “On defending against label flipping attacks on malware detection systems,” *Neural Computing and Applications,* vol. 32, p. 14781–14800, 2020.

[209] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng and S. Yu, "Pdgan: A novel poisoning defense method in federated learning using generative adversarial network," in *International Conference on Algorithms and Architectures for Parallel Processing,* 2019.

[210] G. Tao, X. Chen, Y. Jia, Z. Zhong, S. Ma and X. Zhang, "FIRM: DETECTING ADVERSARIAL AUDIOS BY RECURSIVE FILTERS WITH RANDOMIZATION," in *International Conference on Learning Representations,* 2021.

[211] Y. Qin, N. Carlini, G. Cottrell, I. Goodfellow and C. Raffel, "Imperceptible, robust, and targeted adversarial examples for automatic speech recognition," in *International conference on machine learning,* 2019.

[212] A. Paudice, L. Muñoz-González, A. Gyorgy and E. C. Lupu, "Detection of adversarial training examples in poisoning attacks through anomaly detection," *arXiv preprint arXiv:1802.03041,* 2018.

[213] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security,* 2017.

[214] M. Esmaeilpour, P. Cardinal and A. L. Koerich, "Detection of Adversarial Attacks and Characterization of Adversarial Subspace," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 2020.

[215] H. Kwon, H. Yoon and K.-W. Park, "Acoustic-decoy: Detection of adversarial examples through audio modification on speech recognition system," *Neurocomputing,* vol. 417, p. 357–370, 2020.

[216] Q. Zeng, J. Su, C. Fu, G. Kayas, L. Luo, X. Du, C. C. Tan and J. Wu, "A multiversion programming inspired approach to detecting audio adversarial examples," in *2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN),* 2019.

[217] Y. Wang, J. Qin and W. Wang, "Efficient approximate entity matching using jaro-winkler distance," in *International Conference on Web Information Systems Engineering,* 2017.

[218] Z. Yang, B. Li, P.-Y. Chen and D. Song, "Characterizing audio adversarial examples using temporal dependency," *International Conference on Learning Representations,* 2019.

[219] S. Hussain, P. Neekhara, S. Dubnov, J. McAuley and F. Koushanfar, "{WaveGuard}: Understanding and Mitigating Audio Adversarial Examples," in *30th USENIX Security Symposium (USENIX Security 21),* 2021.

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

[221] S. Chen, X. Huang, Z. He and C. Sun, "DAmageNet: a universal adversarial dataset," *arXiv preprint arXiv:1912.07160,* 2019.

[222] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran and A. Madry, "Adversarial examples are not bugs, they are features," *Advances in neural information processing systems,* vol. 32, 2019.

[223] A. Braunegg, A. Chakraborty, M. Krumdick, N. Lape, S. Leary, K. Manville, E. Merkhofer, L.

Strickhart and M. Walmer, “Apricot: A dataset of physical adversarial attacks on object detection,” in *European Conference on Computer Vision*, 2020.

[224] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt and D. Song, “Natural adversarial examples,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[225] X. Liu, S. Singh, C. Cornelius, C. Busho, M. Tan, A. Paul and J. Martin, “Synthetic Dataset Generation for Adversarial Machine Learning Research,” *arXiv preprint arXiv:2207.10719,* 2022.

[226] A. M. Tripathi and A. Mishra, “Adv-ESC: Adversarial attack datasets for an environmental sound classification,” *Applied Acoustics,* vol. 185, p. 108437, 2022.

[227] H. Ali, M. S. Khan, A. AlGhadhban, M. Alazmi, A. Alzamil, K. Al-Utaibi and J. Qadir, “All Your Fake Detector Are Belong to Us: Evaluating Adversarial Robustness of Fake-news Detectors Under Black-Box Settings,” *IEEE Access,* vol. 9, p. 81678–81692, 2021.

Appendix

Below it has been shared the IDS signatures that were created for the detection of SSJI attacks. The signatures were tested in both Suricata and Snort and are available online[37].

1. alert tcp any any -> any any (msg:"Possible SSJI exploit Python UA"; flow:to_server, established;content:"Python-urllib"; http_header; sid:1000013; classtype: web-application-attack; rev:1;)
2. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST payload HEX Detection"; flow:to_server, established; content:"|41 44 27 29|"; http_client_body; sid:1000011; classtype: web-application-attack; rev:1;)
3. alert tcp any any -> any any (msg:"Possible SSJI exploit POST eval HEX Detection"; flow:to_server, established; content:"|65 76 61 6C|"; http_client_body; sid:1000012; classtype: web-application-attack; rev:1;)
4. alert tcp any any -> any any (msg:"Possible SSJI exploit POST res.end HEX Detection"; flow:to_server, established; content:"|72 65 73 2E 65 6E 64|"; http_client_body; sid:1000015; classtype: web-application-attack; rev:1;)
5. alert tcp any any -> any any (msg:"Possible SSJI exploit POST response.end HEX Detection"; flow:to_server, established; content:"|72 65 73 70 6F 6E 73 65 2E 65 6E 64|"; http_client_body; sid:1000014; classtype: web-application-attack; rev:1;)
6. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET payload HEX Detection"; flow:to_server, established; content:"|41 44 27 29|"; http_uri; sid:1000016; classtype: web-application-attack; rev:1;)
7. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET eval HEX Detection"; flow:to_server, established; content :"|65 76 61 6C|"; http_uri; sid:1000017; classtype: web application-attack; rev:1;)
8. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET res.end HEX Detection"; flow:to_server, established; content:"|72 65 73 2E 65 6E 64|"; http_uri; sid:1000018; classtype: web-application-attack; rev:1;)
9. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET response.end HEX Detection"; flow:to_server, established; content:"|72 65 73 70 6F 6E 73 65 2E 65 6E 64|"; http_uri; sid:1000019; classtype: web-application-attack; rev:1;)
10. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST res.end Detection"; flow:to_server, established; content:"res.end"; nocase; http_client_body; sid:1000023; classtype: web-application-attack; rev:1;)
11. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST response.end Detection"; flow:to_server, established; content:"response.end"; nocase; http_client_body; sid: 1000031; classtype: web-application-attack; rev:1;)
12. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST eval Detection"; flow:to_server, established; content:"eval"; nocase; http_client_body; sid:1000024; classtype: web-application-attack; rev:1;)
13. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET res.end Detection"; flow:to_server, established; content:"res.end"; nocase; http_uri; sid:1000025; classtype web-application-attack; rev:1;)
14. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET response.end Detection"; flow:to_server, established; content:"response.end"; nocase; http_uri; sid:1000026; classtype: web-application-attack; rev:1;)
15. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET eval Detection"; flow:to_server, established; content:" eval"; nocase; http_uri; sid:1000027; classtype: web-application-attack; rev:1;)

[37] https://github.com/esmog/nodexp/blob/master/files/ssji.rules

16. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET res.end URL encoding Detection"; flow:to_server, established; pcre:"/res.end\%\w*\%\d*/"; sid:1000029; classtype: web-application-attack; rev:1;)
17. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET response.end URL encoding Detection"; flow:to_server, established; pcre:"/response.end\%\w*\%\d*/"; sid:1000030; classtype: web-application-attack; rev:1;)
18. alert tcp any any -> any any (msg:"Possible SSJI exploit-GET eval URL encoding Detection"; flow:to_server, established; pcre:"/eval\%\d*\w*\%\w*\%\d*/"; sid:1000028; classtype: web-application-attack; rev:1;)
19. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST eval multiplication detection"; flow:to_server, established; pcre:"/eval\(\d**\d*\)/"; sid:1000010; classtype: web-application-attack; rev:1;)
20. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST eval devision detection"; flow:to_server, established; pcre:"/eval\(\d*\/\d*\)/"; sid:1000032; classtype: web-application-attack; rev:1;)
21. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST res.end random string detection"; pcre:"/res\.end \(.*\)$/"; sid:1000021; classtype: web-application-attack; rev:1;)
22. alert tcp any any -> any any (msg:"Possible SSJI exploit-POST response.end random string detection"; pcre:"/response\.end\(.*\)$/"; sid:1000022; classtype: web-application-attack; rev:1;)