MITIGATING CYBERATTACK WITH MACHINE LEARNING-BASED FEATURE SPACE TRANSFORMS

by

Gbadebo Gbadero Ayoade

APPROVED BY SUPERVISORY COMMITTEE:

Latifur Khan, Co-Chair

Kevin W. Hamlen, Co-Chair

Murat Kantarcioglu

Bhavani Thuraisingham

To my parents, Dr and Mrs. Adedapo Ayoade.

MITIGATING CYBERATTACK WITH MACHINE LEARNING-BASED FEATURE SPACE TRANSFORMS

by

GBADEBO GBADERO AYOADE, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2019

ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Latifur Khan and Dr. Kevin Hamlen for the patience, support and guidance in completing this dissertation.

I would also like to thank my dissertation committee, Dr. Murat Kantarcioglu and Dr. Bhavani Thuraisingham, for their support and encouragement.

I extend my gratitude to my parents, Dr. and Mrs Adedapo Ayoade and my siblings, Adewole Ayoade and Adejonwo Ayoade. This dissertation would not be possible without their patience and sacrifice.

I would like to thank all my friends, Dr. Fred Araujo, Dr. Khaled Al-Naami, Dr. Vishal Karande and Dr. Swarup Chandra for making this a memorable journey.

The research reported herein was supported in part by ONR award N00014-17-1-2995; NSF award No. 1513704; NSA award H98230-15-1-0271; AFOSR award FA9550-14-1-0173; DARPA award FA8750-19-C-0006; NSF under award No. 1054629; AFOSR under award No. FA9550-12-1-0077; ONR awards N00014-14-1-0030; ARO award W911-NF-18-1-0249; an endowment from the Eugene McDermott family; NSF FAIN awards DGE-1931800, OAC-1828467, and DGE-1723602; NSF awards DMS-1737978 and MRI-1828467; an IBM faculty award (Research); and an HP grant. Any opinions, recommendations, or conclusions expressed are those of the authors and not necessarily of the aforementioned supporters.

October 2019

MITIGATING CYBERATTACK WITH MACHINE LEARNING-BASED FEATURE SPACE TRANSFORMS

Gbadebo Gbadero Ayoade, PhD
The University of Texas at Dallas, 2019

Supervising Professors: Latifur Khan, Co-Chair
Kevin W. Hamlen, Co-Chair

With the increase in attacks on software systems, there is a need for a new approach in software defense. In this work, we explore machine learning-based approaches for the detection and mitigation of attacks. A machine learning-based approach can help to learn patterns that can be used to detect future attacks. However, due to limited labeled training data in cyber security domain, machine learning-based approaches perform poorly in attack detection. This work overcomes these challenges by leveraging two main methods. First, we leverage deception-based honey-patching to label the attack data that can then used to train machine learning models to detect future attacks. The second method uses transfer learning method to adapt data from domains with sufficient labels to train our models and test on data from a different domain. By leveraging these methods, we can show an increase in the detection performance of machine learning-based models in cyber attack defense systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION [1] [2] [3] [4] [5]

Cyberdeceptive defenses are increasingly vital for protecting organizational and national critical infrastructures from asymmetric cyber threats. Market forecasts predict an over $2 billion industry for cyberdeceptive products by 2022 (Mordor Intelligence, 2018), including major product releases by Rapid7, TrapX, LogRhythm, Attivo, Illusive Networks, Cymmetria, and many others in recent years (Sadowski and Kau, 2019).

These new defense layers are rising in importance because they enhance conventional defenses by shifting asymmetries that traditionally burden defenders back on attackers. For example, while conventional defenses invite adversaries to find just one critical vulnerability to successfully penetrate the network, deceptive defenses challenge adversaries to discern which vulnerabilities among a sea of apparent vulnerabilities (many of them traps) are real.

---

[1] This chapter contains material previously published as: Gbadebo Ayoade, Frederico Araujo, Khaled Al-Naami, Ahmad M. Mustafa, Yang Gao, Kevin W. Hamlen, and Latifur Khan. "Automating Cyberdeception Evaluation with Deep Learning." In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS)*, January 2020. Ayoade led the data generation, applied deep learning techniques aspect of this research

[2] This chapter contains material previously published as: Frederico Araujo, Gbadebo Ayoade, Khaled Al-Naami, Yang Gao, Kevin W. Hamlen, and Latifur Khan. "Improving Intrusion Detectors by Crook-sourcing." In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC)*, December 2019. Ayoade led the data generation, applied deep learning techniques aspects of this research

[3] This chapter contains material previously published as: Ehab Al-Shaer, Jinpeng Wei, Kevin W. Hamlen, Cliff Wang. "Chapter 8 Deception-Enhanced Threat Sensing for Resilient Intrusion Detection", *Springer Science and Business Media LLC, 2019* Ayoade led the data generation aspects of this research

[4] This chapter contains material previously published as: Li, Yifan., Yang. Gao, Gbadebo Ayoade, Hemeng Tao, Latifur Khan, and Bhavani Thuraisingham. "Multistream classification for cyber threat data with heterogeneous feature space." In *The World Wide Web Conference, WWW '19*, New York, NY, USA, pp. 2992–2998. ACM. Ayoade led the cyberthreat detection aspect of this research

[5] This chapter contains material previously published as: Gbadebo Ayoade, Swarup Chandra, Latifur Khan, Kevin Hamlen, and Bhavani Thuraisingham. "Automated Threat Report Classification over Multi-source Data." *In Proceedings of the 4th IEEE International Conference on Collaboration and Internet Computing (CIC)*, pp. 236–245, October 2018. Ayoade led the machine learning aspect including the design, implementation and evaluation of the approaches used in this research.

As attacker-defender asymmetries increase with the increasing complexity of networks and software, deceptive strategies for leveling those asymmetries will become increasingly essential for scalable defense.

Robust evaluation methodologies are a critical step in the development of effective cyberdeceptions; however, cyberdeception evaluation is frequently impeded by the difficulty of conducting experiments with appropriate human subjects. Capturing the diversity, ingenuity, and resourcefulness of real APTs tends to require enormous sample sizes of rare humans having exceptional skills and expertise. Human deception research raises many ethical dilemmas that can lead to long, difficult approval processes (Baumrind, 1979). Even when these obstacles are surmounted, such studies are extremely difficult to replicate (and therefore to confirm), and results are often difficult to interpret given the relatively unconstrained, variable environments that are the contexts of real-world attacks.

Progress in cyberdeceptive defense hence demands efficient methods of conducting preliminary yet meaningful evaluations without humans in the loop. Human subject evaluation can then be reserved as a final, high-effort validation of the most promising, mature solutions.

Toward this goal, this thesis proposes and critiques a machine learning-based approach for evaluating cyberdeceptive software defenses without human subjects. Although it is extremely difficult to emulate human decision-making automatically for synthesizing attacks, our approach capitalizes on the observation that in practice cyber attackers rely heavily upon mechanized tools for offense. For example, human bot masters rely primarily upon reports delivered by automated bots to assess attack status and reconnoiter targets, and they submit relatively simple commands to the botnet to carry out complex killchains that are largely mechanized as malicious software. In such scenarios, deceiving the mechanized agents goes a long way toward deceiving their human masters. Automating the machine-versus-machine part of the deception evaluation is therefore both feasible and useful.

A major approach to implementing cyber-deception is through honey-patching. For example, *honey-patches* (Araujo et al., 2014, 2015, 2019; Araujo and Hamlen, 2015) introduce

application layer deceptions by selectively replacing software security patches with decoy vulnerabilities. Attempted exploits transparently redirect the attacker's session to a decoy environment where the exploit is allowed to succeed. This allows the system to observe subsequent phases of the attacker's killchain without risk to genuine assets.

We therefore propose an evaluation methodology that leverages machine learning to (1) generate realistic streams of synthetic traffic comprised of benign interactions and attacks based on real threat data and vulnerabilities, and (2) automatically adapt the synthetic traffic in an effort to evade observed (possibly deceptive) responses to the attacks. The goal is to obtain the maximum evaluative power of adaptive deceptive defenses without explicit human adversarial engagement.[6] As a case study, we apply our technique to evaluate a network-level intrusion detection system (IDS) equipped with embedded honeypots at the application layer.

1.1 Cyberthreat Detection with Domain Adaptation

Due to the scarcity of labeled data, we develop a domain adaptation framework to address the challenge of using data from multiple domains to train a classifier in detecting cyber-attacks. For example, many organizations are prone to cyber-attacks. These organizations deploy threat monitoring tools that collect network packet and system call events as data streams. To detect these attacks, we have to overcome the challenge of linking these low-level traces to attacker tactics and intent.

The scarcity of labeled data introduces more limitations in deploying machine learning techniques in detecting cyber-attacks. Data from some domains are well labeled and data from these domains can be used to train classifiers from other domains. However, there is a limitation of adapting data from different domains to train a machine learning model.

[6]The implementation and datasets used in this dissertation are available in https://github.com/cyberdeception/deepdig.

For example, data streams obtained from Linux machines cannot be used to predict attack events on a host with a Windows operating system. This is due to the difference in the way events are recorded on both systems.

Another challenge is in detecting malicious products being sold on the dark web. Due to the various platforms available for malicious actors to sell illegal goods, such as stolen credit card numbers, social security numbers, malicious software, root-kits, etc, we need a system that can obtain labeled train data from different dark web domains and use them to extract products sold on other dark web domains. By leveraging domain adaptation, we can increase the performance of the product extraction technique.

Our proposed framework incorporates transfer learning techniques (Wei et al., 2016), and improve the overall classification accuracy by constructing an objective function to find an optimal latent feature space for both source and target data, which also preserve the structure of both data sets.

Our goal is to determine high level concepts of attacker tactics from low level network and system call traces. Our approach begins with the collection of network and system level call traces from attack executions. Second, we extract features such as packet length, packet size, packet direction and packet timestamps from packet data and n-gram features from the system call traces. Third, we apply domain adaptation if the training and the testing data originate from different domains.

1.2 Automated Threat Report Classification Over Multi-Source Data

With an increase in targeted attacks such as advanced persistent threats (APTs), enterprise system defenders require comprehensive frameworks that allow them to collaborate and evaluate their defense systems against such attacks. MITRE has developed a framework which includes a database of different kill-chains, tactics, techniques, and procedures that attackers employ to perform these attacks.

In this work, we leverage natural language processing techniques to extract attacker actions from threat report documents generated by different organizations and automatically classify them into standardized tactics and techniques, while providing relevant mitigation advisories for each attack. A naïve method to achieve this is by training a machine learning model to predict labels that associate the reports with relevant categories. In practice, however, sufficient labeled data for model training is not always readily available, so that training and test data come from different sources, resulting in a bias. A naïve model would typically underperform in such a situation. We address this major challenge by incorporating an importance weighting scheme called bias correction that efficiently utilizes available labeled data, given threat reports, whose categories are to be automatically predicted. We empirically evaluated our approach on 18,257 real-world threat reports generated between the years 2000 and 2018 from various computer security organizations to demonstrate its superiority by comparing its performance with an existing approach.

1.3 Contribution of this dissertation

In summary, the contribution of this work is as follows.

1.3.1 Cyber deception based defenses

- We present the design of a framework for replay and generation of web traffic that statistically mutates and injects scripted attacks into the output streams to more effectively train, test, and evaluate deceptive, concept-learning IDSes.
- We evaluate our approach on large-scale network and system events gathered via simulation over a test bed built atop production web software, including the Apache web server, OpenSSL, and PHP.
- We propose an adaptive deception detector to cope with adaptive defenses to detect outliers in the presence of concept-evolving data streams.

1.3.2 Cyber attack detection using domain adaptation

- We build a framework that provides an end-to-end system consisting of data collection, prepossessing, and feature extraction.

- We develop a novel domain adaptation technique with a novel objective function that leverages data from different domain environments. For example, we train a classifier on trace data collected from Windows operating system environment and test on trace data collected from Linux operating system environment to predict attacker tactics.

1.3.3 For threat report classification

- We build a framework that provides an end-to-end system consisting of data collection, prepossessing, and feature extraction steps applied to 18,257 threat reports generated in the last 18 years from reputable security organizations.

- Using machine learning techniques, we address the challenges of non-standard report formats, threat categories, and limited labeled data availability, for categorizing reports and generating an appropriate actionable response to threat descriptions.

- We empirically evaluate our framework on a large number of real-world threat reports, and show the effectiveness of our approach, i.e., we observe an improvement of up to 78% in classification accuracy compared to the existing approach.

1.4 Outline of the dissertation

The rest of the dissertation is organized as follows. Chapter 2 gives a background on the approaches used in this dissertation.

Chapter 3 discusses our approach and evaluaton of leveraging cyber deception based techniques in mitigating cyber attacks. Chapter 4 discusses our approach on using domain

adaption for cyber attack detection. Chapter 5 outline our approach of classifying attack reports for faster discovery for use by cyber security defense teams.

Chapter 6 discusses our approach of leveraging blockchain for managing IoT data privacy for everyday users Chapter 7 gives a summary of this dissertation and future work and Appendix A presents other works in the protection of vulnerable smart contracts from attack exploitation.

CHAPTER 2

BACKGROUND [1] [2] [3] [4] [5]

In this chapter we present the challenges encountered in building machine learning based intrusion detection systems. We also discuss relevant background information on existing and previous studies.

2.0.1 Challenges in IDS Evaluation

One of the major challenges for evaluation of deceptive IDSes is the general inadequacy of static attack datasets, which cannot react to deceptive interactions. Testing deceptive defenses with these datasets renders the deceptions useless, missing their value against reactive threats.

---

[1] This chapter contains material previously published as: Gbadebo Ayoade, Frederico Araujo, Khaled Al-Naami, Ahmad M. Mustafa, Yang Gao, Kevin W. Hamlen, and Latifur Khan. "Automating Cyberdeception Evaluation with Deep Learning." In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS)*, January 2020. Ayoade led the data generation, applied deep learning techniques aspects of this research

[2] This chapter contains material previously published as: Frederico Araujo, Gbadebo Ayoade, Khaled Al-Naami, Yang Gao, Kevin W. Hamlen, and Latifur Khan. "Improving Intrusion Detectors by Crook-sourcing." In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC)*, December 2019. Ayoade led the data generation, applied deep learning techniques aspects of this research

[3] This chapter contains material previously published as: Ehab Al-Shaer, Jinpeng Wei, Kevin W. Hamlen, Cliff Wang. "Chapter 8 Deception-Enhanced Threat Sensing for Resilient Intrusion Detection", *Springer Science and Business Media LLC, 2019* Ayoade led the data generation aspects of this research

[4] This chapter contains material previously published as: Li, Yifan., Yang. Gao, Gbadebo Ayoade, Hemeng Tao, Latifur Khan, and Bhavani Thuraisingham. "Multistream classification for cyber threat data with heterogeneous feature space." In *The World Wide Web Conference, WWW '19*, New York, NY, USA, pp. 2992–2998. ACM. Ayoade led the cyberthreat detection aspect of this research

[5] This chapter contains material previously published as: Gbadebo Ayoade, Swarup Chandra, Latifur Khan, Kevin Hamlen, and Bhavani Thuraisingham. "Automated Threat Report Classification over Multi-source Data." *In Proceedings of the 4th IEEE International Conference on Collaboration and Internet Computing (CIC)*, pp. 236–245, October 2018. Ayoade led the machine learning aspect including the design, implementation and evaluation of the approaches used in this research.

To mitigate this problem, a method of dynamic attack synthesis is required. A suitable solution must learn a model of how adversarial agents are likely to react based on their reactions to similar feedback during real-world interactions mined from real attack data. The accuracy of such predictions depends upon the complexity of deceptive responses and the decision logic of the adversaries. For example, when defensive responses are binary (*viz.* accept or reject) or a finite list of error messages, accurate prediction is more feasible than when the output space is large (e.g., arbitrary textual messages). Likewise, automated agents tend to have high predictability (e.g., learnable by emulating their software logic on desired inputs), whereas human agents are far more difficult to predict.

2.0.2 Intrusion detection datasets

Another challenge in evaluating intrusion detection systems is the availability of data sets that represent real life patterns. Due to privacy concerns, Internet service providers do not provide access to traffic data through their network. This has given rise to researchers generating their own datasets which could be either biased.

According to the survey paper by (Bhuyan et al., 2014), they classified the available datasets under 3 categories which include synthetic, benchmark and real life dataset. One of the recognized challenges in generating dataset is how to capture as many variations and properties in different system set up, for example dataset from a Linux and Windows environment will be different even across the same Operating system versions or distributions. Some of the available dataset are very old (Ahmed et al., 2016), e.g DARPA2000 which do not represent today's traffic or attack scenarios.

In order to avoid this pitfalls, we developed a synthetic dataset generator that is inspired by Windtunnel system (Boggs et al., 2014). Using similar approach, we statistically generate benign and attack traffic and we collected both network and system level trace for our evaluation. The technique allows us to introduce randomness therefore, creating a dataset

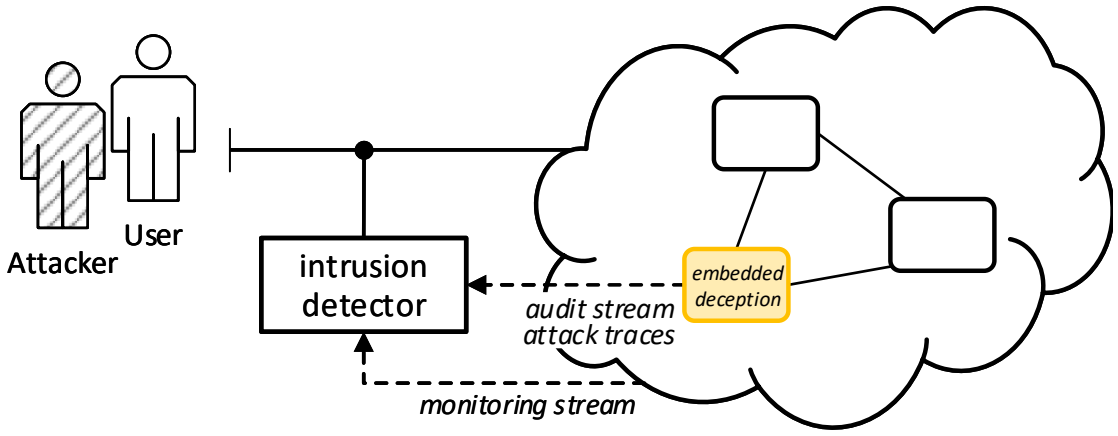


Figure 2.1: Deceptive IDS training overview

that is both synthetic and also close to real life dataset traffic. Our dataset also contains both the network and trace and the corresponding host level system traces, which allows us to use both external and internal system events for intrusion detection.

Our data generation framework differs from Windtunnel, because we can generate both encrypted and un-encrypted network traffic. Our monitors is also enhanced since we can gather more attack information using the deception based framework - honeypatched servers as demonstrated in this dissertation.

2.0.3 Deception-enhanced Intrusion Detection

Our evaluation approach targets IDS defenses enhanced with deceptive attack-responses (e.g., (Araujo et al., 2014; Avery and Spafford, 2017; Crane et al., 2013)). Figure 2.1 depicts the general approach. Unlike conventional intrusion detection, deception-enhanced IDSes incrementally build a model of *legitimate* and *malicious* behavior based on audit streams and attack traces collected from successful deceptions. The deceptions leverage user interactions at the network, endpoint, or application layers to solicit extra communication with adversaries and waste their resources, misdirect them, or gather intelligence. This augments the classifier with security-relevant feature extraction capabilities not available to typical network intrusion detectors.

2.1 Online Adaptive Metric Learning

OAML (Gao et al., 2019) is a recently advanced deep learning approach that improves instance separation by transforming input features to a new latent space. This generates a new latent feature space where similar instances are closer together and dissimilar instances are separated farther. It extends online similarity metric learning (OML) (Li et al., 2018; Chechik et al., 2010; Jain et al., 2008; Jin et al., 2009; Breen et al., 2002), which employs *pairwise* and *triplet* constraints: A pairwise constraint takes two dissimilar/similar instances, while a triplet constraint (A, B, C) combines similar instances A and B with a dissimilar instance C.

We choose OAML since non-adaptive OML usually learns a pre-selected linear metric (e.g., Mahalanobis distance (Xiang et al., 2008)) that lacks the complexity to learn non-linear semantic similarities among class instances, which are prevalent in intrusion detection scenarios. In addition, using a non-adaptive method results in a fixed metric which suffers from bias to a specific dataset. OAML overcomes these disadvantages by adapting its metric learning model to accommodate more constraints in the observed data. Its metric function learns a dynamic latent space from the Bi-Di and N-Gram feature spaces, which can include both linear and highly non-linear functions.

2.2 Intrusion Detection

2.2.1 ML-based Intrusion Detection

Machine learning-based IDSes (cf., (Garcia-Teodoro et al., 2009; Chandola et al., 2009; Patcha and Park, 2007; Masud et al., 2008, 2010)) find patterns that do not conform to expected system behavior, and are typically classified into *host-based* and *network-based* approaches.

Host-based detectors recognize intrusions in the form of anomalous system call trace sequences, in which co-occurrence of events is key to characterizing malicious behavior. For example, malware activity and privilege escalation often manifest specific system call patterns (Chandola et al., 2009). Seminal work in this area has analogized intrusion detection via statistical profiling of system events to the human immune system (Forrest et al., 1996; Hofmeyr et al., 1998). This has been followed by a number of related approaches using histograms to construct profiles of normal behavior (Marceau, 2001). Another frequently-used approach employs a *sliding window* classifier to map sequences of events into individual output values (Warrender et al., 1999; Cohen, 1995a), converting sequential learning into a classic machine learning problem. More recently, long call sequences have been studied to detect attacks buried in long execution paths (Shu et al., 2015).

Network-based approaches detect intrusions using network data. Since such systems are typically deployed at the network perimeter, they are designed to find patterns resulting from attacks launched by external threats, such as attempted disruption or unauthorized access (Bhuyan et al., 2014). Network intrusion detection has been extensively studied in the literature (cf., (Bhuyan et al., 2014; Ahmed et al., 2016)). Major approaches can be grouped into classification-based (e.g., SVM (Eskin et al., 2002), (Awad et al., 2004), Bayesian network (Kruegel et al., 2003)), information-theoretic (Lee and Xiang, 2001), and statistical (Krügel et al., 2002; Kruegel and Vigna, 2003; Kruegel et al., 2005) techniques.

Network-based intrusion detection systems can monitor a large number of hosts at relatively low cost, but they are usually opaque to *local* or *encrypted* attacks. On the other hand, intrusion detection systems operating at the host level have complete visibility of malicious events, despite encrypted network payloads and obfuscation mechanisms (Kim et al., 2007). Our approach therefore complements existing techniques and incorporates host- and network-based features to offer protective capabilities that can resist attacker evasion strategies and detect malicious activity bound to different layers of the software stack.

Another related area of research is *web-based* malware detection that identifies drive-by-download attacks using static analysis, dynamic analysis, and machine learning (Kapravelos et al., 2013; Canali et al., 2011). In addition, other studies focus on *flow-based* malware detection by extracting features from proxy-logs and using machine learning (Bartos et al., 2016).

2.2.2 Feature Extraction for Intrusion Detection

A variety of feature extraction and classification techniques have been proposed to perform host- and network-based anomaly detection (Masud et al., 2011) (Masud et al., 2011). Extracting features from encrypted network packets has been intensively studied in the domain of *website fingerprinting*, where attackers attempt to discern which websites are visited by victims. Users typically use anonymous networks, such as Tor, to hide their destination websites (Wang et al., 2014). However, attackers can often predict destinations by training classifiers directly on encrypted packets (e.g., packet headers only). Relevant features typically include packet length and direction, summarized as a histogram feature vector. HTML markers, percentage of incoming and outgoing packets, bursts, bandwidth, and website upload time have also been used (Panchenko et al., 2011; Dyer et al., 2012). Packet-word vector approaches additionally leverage natural language processing and vector space models to convert packets to word features for improved classification (Alnaami et al., 2015).

Bi-Di leverages packet and uni-burst data and introduces bi-directional bursting features for better classification of network streams. On unencrypted data, host-based systems have additionally extracted features from co-occurrences and sequences of system events, such as system calls (Cabrera et al., 2001; Marceau, 2001). DeepDig uses a hybrid scheme that combines both host- and network-based approaches via a modified ensemble technique.

2.3 Domain Adaption

In machine learning, training data and test data are usually assumed to be generated from the same domain and belong to the same data distribution, which may not be the case. (Zadrozny, 2004). For example, data traces collected from machines running Linux exhibit different system call events than machines running Windows. Feature extraction from these traces will yield different feature dimensions and distributions. This challenge limits the use of data from across different domains and compounds the problem of data availability for training a classifier. The differences in domain can be based on two aspects which includes distinct number of features and distinct feature distribution.

CHAPTER 3

DEEPDIG: AUTOMATING CYBERDECEPTION EVALUATION WITH DEEP LEARNING [1] [2] [3] [4]

3.1 Approach Overview

In this chapter, we present our approach to quantitatively assess the resiliency of adaptive, deceptive, concept-learning defenses for web services against adaptive adversaries (Araujo et al., 2019) (Ayoade et al., 2020). Our approach therefore differs from works that measure only absolute IDS accuracy. We first present our approach for generating web traffic to replay normal and malicious user behavior, which we harness to automatically generate training and test datasets for attack classification (§3.1.1). We then discuss the testing harness and analysis used to investigate the effects of different attack classes and varying numbers of attack instances on the predictive power and accuracy of intrusion detection (§3.1.2).

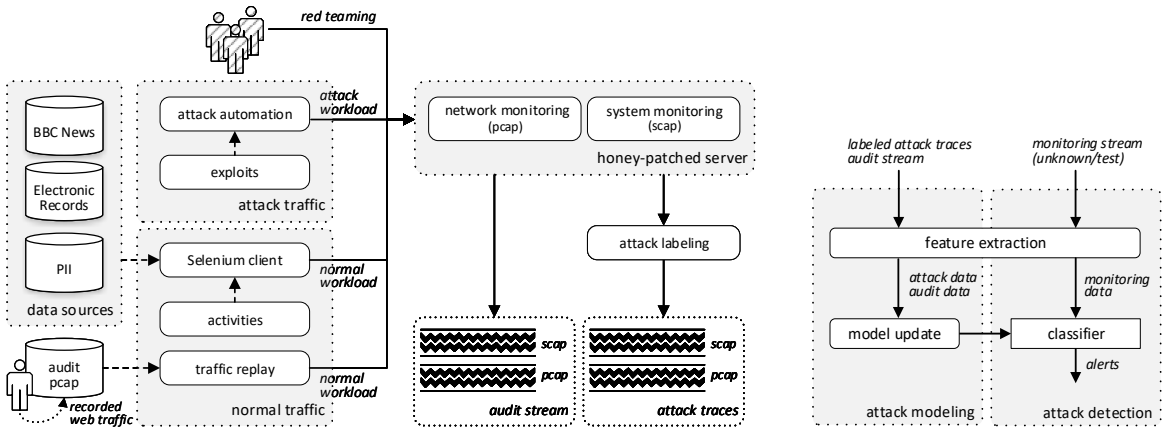
---

[1] This chapter contains material previously published as: Gbadebo Ayoade, Frederico Araujo, Khaled Al-Naami, Ahmad M. Mustafa, Yang Gao, Kevin W. Hamlen, and Latifur Khan. "Automating Cyberdeception Evaluation with Deep Learning." In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS)*, January 2020. Ayoade led the data generation, applied deep learning techniques aspects of this research

[2] This chapter contains material previously published as: Frederico Araujo, Gbadebo Ayoade, Khaled Al-Naami, Yang Gao, Kevin W. Hamlen, and Latifur Khan. "Improving Intrusion Detectors by Crook-sourcing." In *Proceedings of the 35th Annual Computer Security Applications Conference (ACSAC)*, December 2019. Ayoade led the data generation, applied deep learning techniques aspects of this research

[3] This chapter contains material previously published as: Ehab Al-Shaer, Jinpeng Wei, Kevin W. Hamlen, Cliff Wang. "Chapter 8 Deception-Enhanced Threat Sensing for Resilient Intrusion Detection", *Springer Science and Business Media LLC, 2019* Ayoade led cyberthreat data generation aspects of this research

[4] This chapter contains material previously published as: Ahmad M. Mustafa, Gbadebo Ayoade, Khaled Al-Naami, Latifur Khan, Kevin W. Hamlen, Bhavani Thuraisingham, Frederico Araujo. "Unsupervised deep embedding for novel class detection over data stream", In *Proceedings of the 2017 IEEE International Conference on Big Data (Big Data)*, 2017 Ayoade led cyberthreat data generation aspects of this research



(a) traffic analysis (b) data analysis

Figure 3.1: Overview of (a) automated workload generation for cyberdeception evaluation, and (b) deceptive IDS training and testing.

3.1.1 Traffic Analysis

Our evaluation methodology seeks to create realistic, end-to-end workloads and attack killchains to functionally test cyberdeceptive defenses embedded in commodity server applications and process decoy telemetry for feature extraction and IDS model evolution. Figure 3.1a shows an overview of our traffic generation framework. It streams *encrypted* legitimate and malicious workloads onto endpoints enhanced with embedded deceptions, resulting in labeled audit streams and attack traces (collected at decoys) for training set generation.

Workload Generation. Rather than evaluating deception-enhanced IDSes with existing, publicly available intrusion datasets (which are inadequate for the reasons outlined in §2.0.1), our evaluation interleaves attack and normal traffic following prior work on defense-in-depth (Boggs et al., 2014; Araujo et al., 2019), and injects benign payloads as data into attack packets to mimic evasive attack behavior. The generated traffic contains attack payloads against realistic exploits (e.g., weaponizing recent CVEs for reconnaissance and initial infection), and our framework automatically extracts labeled features from the monitoring network and system traces to (re-)train the classifiers.

Legitimate workload. The framework uses both *real* user sessions and *automated simulation* of various user actions to compose legitimate traffic. Real interactions comprise web traffic that is monitored and recorded as *audit pcap* data in the targeted operational environment (e.g., regular users in a local area network). The recorded sessions are replayed by our framework and streamed as normal workload onto endpoints embedding deceptions.

These regular data streams are enriched with simulated interactions, which are created by automating complex user actions on typical web application, leveraging *Selenium* (Selenium, 2019) to automate user interaction with web applications (e.g., clicking buttons, filling out forms, navigating a web page). To create realistic workloads, our framework feeds from online data sources, such as the BBC text corpus (Greene and Cunningham, 2006), online text generators (Mockaroo, 2018) for personally identifiable information (e.g., usernames, passwords), and product names to populate web forms. To ensure diversity, we statistically sample the data sources to obtain user input values and dynamically generate web content. For example, blog title and body are statistically sampled from the BBC text corpus, while product names are picked from the product names data source.

Our implementation defines different customizable user activities that can be repeated with varying data feeds and scheduled to simulate different workload profiles and temporal patterns. These include web page browsing, e-commerce website navigation, blog posting, and interacting with a social media web application. The setup includes common web software stacks, such as CGI web applications and PHP-based Wordpress applications hosted on a monitored Apache web server as shown in Table 3.2.

Attack workload. Attack traffic is generated based on real vulnerabilities. The procedure harnesses a collection of scripted attacks (crafted using Bash, Python, Perl, or Metasploit scripts) to inject malicious client traffic against endpoints in the tested environment. Attacks can be easily extended and tailored to specific test scenarios during evaluation design, without modifications to the framework, which automates and schedules attacks according

Table 3.1: Summary of attack workload

#	Attack Type	Description	Software
1	CVE-2014-0160	Information leak	Openssl
2	CVE-2012-1823	System remote hijack	PHP
3	CVE-2011-3368	Port scanning	Apache
4–10	CVE-2014-6271	System hijack (7 variants)	Bash
11	CVE-2014-6271	Remote Password file read	Bash
12	CVE-2014-6271	Remote root directory read	Bash
13	CVE-2014-0224	Session hijack and information leak	Openssl
14	CVE-2010-0740	DoS via NULL pointer dereference	Openssl
15	CVE-2010-1452	DoS via request that lacks a path	Apache
16	CVE-2016-7054	DoS via heap buffer overflow	Openssl
17–22	CVE-2017-5941*	System hijack (6 variants)	Node.js

*used for testing only, as n-day vulnerability.

Table 3.2: Summary of normal and attack workloads

Normal workload summary		
Activity	**Application**	**Description**
Post	CGI web app	Posting blog on a guestbook CGI web application
Post	Wordpress	Posting blog on wordpress
Post	Wordpress buddypress plugin	Posting comment on social media web application
Registration	Wordpress woocommerce plugin	Product registration and product description
Ecommerce	Wordpress woocommerce plugin	Ordering of a product and checkout
Browse	Wordpress	Browsing through a blog post
Browse	Wordpress buddypress	Browsing through a social media page
Browse	Wordpress woocommerce plugin	Browsing product catalog
Registration	Wordpress	User registration
Registration	Wordpress woocommerce plugin	Coupon registration

to parametric statistical models defined by the targeted evaluation (e.g., prior probability of an attack, attack rates, reconnaissance pattern).

In the case study reported in §3.2, multiple exploits for recent CVEs were scripted to carry out different malicious activities (i.e., different attack payloads), such as leaking password files and invoking shells on the remote web server. These vulnerabilities are important as attack vectors because they range from sensitive data exfiltration to complete control and remote code execution. The post-infection payloads execute tasks such as tool acquisition, basic environment reconnaissance (e.g., active scanning with Nmap, passive inspection of system logs), password file access, root certificate exfiltration, and attempts at gaining access to other machines in the network.

Monitoring & Threat Data Collection. Our framework tracks two lifecycle events associated with monitored decoys: upon a *decoy hit*, the framework records the timestamp that denotes the beginning of an attack session (i.e., when a security condition is met). After the corresponding *abort* event arrives (i.e., session disconnection), the monitoring component extracts the session trace (delimited by the two events), labels it, and stores the trace outside the decoy for subsequent feature extraction. Since the embedded deceptions should only host attack sessions, precisely collecting and labeling their traces (at both the network and OS level) is effortless using this strategy.

Our approach distinguishes between three separate input data streams: (1) the *audit stream*, collected at the target honey-patched server, (2) *attack traces*, collected at decoys, and (3) the *monitoring stream*, the actual test stream collected from regular servers. Each of these streams contains network packets and OS events captured at each server environment. To minimize performance impact, we use two powerful and efficient software monitors: *sysdig* (to track system calls and modifications made to the file system), and *tcpdump* (to monitor ingress and egress of network packets). Specifically, monitored data is stored outside decoy environments to avoid possible tampering with collected data.

Our monitoring and data collection solution is designed to scale for large, distributed on-premise and cloud deployments. The host-level telemetry leverages a mainstream kernel module (Sysdig, 2019) that implements non-blocking event collection and memory-mapped event buffer handling for minimal computational overhead. This architecture allows system events to be safely collected (without system call interposition) and compressed by a containerized user space agent that is oblivious to other objects and resources in the host environment. The event data streams originated from the monitored hosts are exported to a high-performance, distributed S3-compatible object storage server (MinIO, 2019), designed for large-scale data infrastructures.

3.1.2 Data Analysis

Using the continuous audit stream and incoming attack traces as labeled input data, our approach enables concept-learning IDSes to incrementally build supervised models that are able to capture legitimate and malicious behavior. As illustrated in Figure 3.1b, the raw training set (composed of both audit stream and attack traces) is piped into a feature extraction component that selects relevant, non-redundant features and outputs feature vectors—*audit data* and *attack data*—that are grouped and queued for subsequent model update. Since the initial data streams are labeled and have been preprocessed, feature extraction becomes very efficient and can be performed automatically. This process repeats periodically according to an administrator-specified policy.

Network Packet Analysis. Each packet transmitted and received forms the basic unit of information flow for our packet-level analysis. *Bidirectional* (Bi-Di) (Al-Naami et al., 2016) (Al-Naami et al., 2019) (Al-Shaer et al., 2019) features are extracted from the patterns observed on this network data. Due to encrypted network traffic opacity, features are extracted from TCP packet headers. Packet data length and transmission time are extracted from network sessions. We extract histograms of packet lengths, time intervals, and directions. To reduce the dimension of the generated features, we apply *bucketization* to group TCP packets into correlation sets based on frequency of occurrence.

Uni-burst features include burst size, time, and count of groups of packets transmitted consecutively in one TCP window. *Bi-burst features* include time and size attributes of consecutive groups of packets transmitted in two consecutive TCP windows.

System Call Analysis. In order to capture events from within the host, we extract features from system-level OS events. Event types include *open*, *read*, *select*, etc., with the corresponding process name. Leveraging N-Gram feature extraction, we build a histogram of the N-Gram occurrences. N-Gram is a contiguous sequences of system call events. We

consider four types of such N-Gram: *uni-events*, *bi-events*, *tri-events*, and *quad-events* are sequences of 1–4 consecutive system call events (respectively).

3.1.3 Classification

Ensemble SVM. After feature extraction, we leverage SVM to classify both Bi-DI and N-Gram features. SVM uses a convex optimization approach by mapping non-linearly separated data to a higher dimensional linearly distinguishing space. With the new linearly separable space, SVM can separate positive (attack) and negative (benign) training instances by a hyperplane with the maximum gap possible. Prediction is assigned based on which side of the hyperplane an instance resides.

The models built from Bi-Di and N-Gram are combined into an ensemble to obtain a better predictive model. Rather than concatenating the features from both Bi-Di and N-Gram, which has the drawback of introducing normalization issues, the ensemble combines multiple classifiers to obtain a better outcome by majority voting. In our case, for each classification output by the classifier models, we obtain the predicted label and the confidence probability of each of the individual classifiers. The outcome of the classifier with the maximum confidence is picked for the predicted instance.

Confidence is rated using Platt scaling (Platt, 1999), which uses the following sigmoid-like function to compute the classification confidence:

$$P(y = 1|x) = \frac{1}{1 + \exp\left(Af(x) + B\right)} \tag{3.1}$$

where y is the label, x is the testing vector, $f(x)$ is the SVM output, and A and B are scalar parameters learned using Maximum Likelihood Estimation (MLE). This yields a probability measure of how much a classifier is confident about assigning a label to a testing point.

Online Adaptive Metric Learning.

OAML leverages artificial neural networks (ANNs) which consist of a set of hidden layers where the output is fed as input to an independent metric-embedding layer (MEL). The

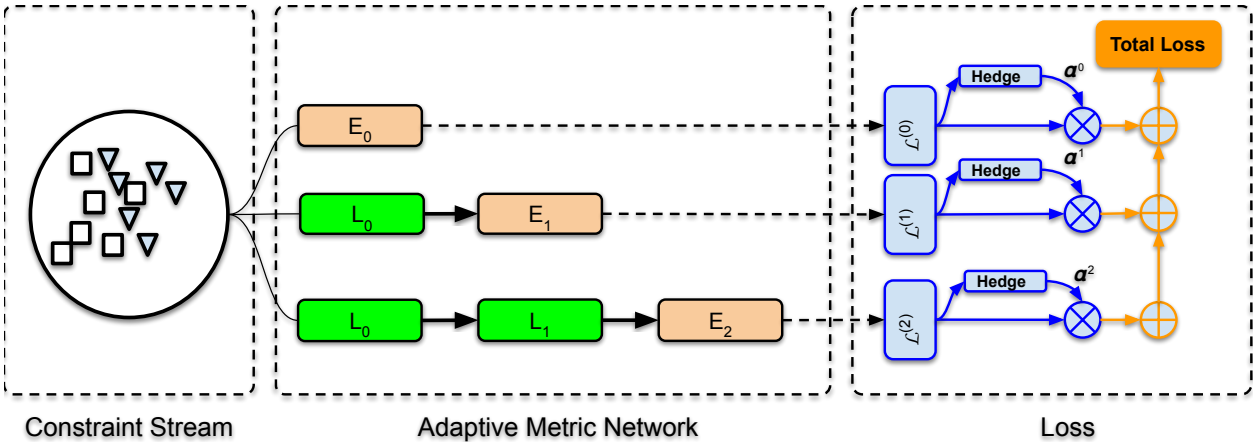


Figure 3.2: OAML network structure. Each layer L_i is a linear transformation output to a rectified linear unit (ReLU) activation. Embedding layers E_i connect input or hidden layers. Linear model E_0 maps the input feature space to the embedding space.

MELs output an n-dimensional vector in an embedded space that clusters similar instances. The importance of model generated by each MEL layer is determined by a metric weight assigned to each MEL. The output of this embedding is used as input to a k-NN classifier, as detailed below.

Problem Setting. Let $S = \{(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-})\}_{t=1}^{T}$ be a sequence of triplet constraints sampled from the data, where $\{\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-}\} \in \mathcal{R}^d$, and $\boldsymbol{x_t}$ (anchor) is similar to $\boldsymbol{x_t^+}$ (positive) but dissimilar to $\boldsymbol{x_t^-}$ (negative). The goal of OAML is to learn a model $\boldsymbol{F} : \mathcal{R}^d \mapsto \mathcal{R}^{d'}$ such that $||\boldsymbol{F}(\boldsymbol{x_t}) - \boldsymbol{F}(\boldsymbol{x_t^+})||_2 \ll ||\boldsymbol{F}(\boldsymbol{x_t}) - \boldsymbol{F}(\boldsymbol{x_t^-})||_2$. Given these parameters, the objective is to learn a metric model with adaptive complexity while satisfying the constraints. The complexity of $\boldsymbol{F}$ must be adaptive so that its hypothesis space is automatically modified.

Overview. Consider a neural network with L hidden layers, where the input layer and the hidden layer are connected to an independent MEL. Each embedding layer learns a latent space where similar instances are clustered and dissimilar instances are separated.

Figure 3.2 illustrates our ANN. Let $E_\ell \in \{E_0, \ldots, E_L\}$ denote the ℓ^{th} metric model in OAML (i.e., the network branch from the input layer to the ℓ^{th} MEL). The simplest OAML model E_0 represents a linear transformation from the input feature space to the

metric embedding space. A weight $\alpha^{(\ell)} \in [0, 1]$ is assigned to E_ℓ, measuring its importance in OAML.

For a triplet constraint $(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-})$ that arrives at time t, its metric embedding $f^{(\ell)}(\boldsymbol{x_t^*})$ generated by E_ℓ is

$$f^{(\ell)}(\boldsymbol{x_t^*}) = h^{(\ell)}\Theta^{(\ell)} \tag{3.2}$$

where $h^{(\ell)} = \sigma(W^{(\ell)}h^{(\ell-1)})$, with $\ell \geq 1$, $\ell \in \mathbb{N}$, and $h^{(0)} = \boldsymbol{x_t^*}$. Here $\boldsymbol{x_t^*}$ denotes any anchor $\boldsymbol{x_t}$ (positive $\boldsymbol{x_t^+}$ or negative $\boldsymbol{x_t^-}$ instance), and $h^{(\ell)}$ is the activation of the ℓ^{th} hidden layer. Learned metric embedding $f^{(\ell)}(\boldsymbol{x_t^*})$ is limited to a unit sphere (i.e., $||f^{(\ell)}(\boldsymbol{x_t^*})||_2 = 1$) to reduce the search space and accelerate training.

During the training phase, for every arriving triplet $(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-})$, we first retrieve the metric embedding $f^{(\ell)}(\boldsymbol{x_t^*})$ from the ℓ^{th} metric model using Eq. 3.2. A local loss $\mathcal{L}^{(\ell)}$ for E_ℓ is evaluated by calculating the similarity and dissimilarity errors based on $f^{(\ell)}(\boldsymbol{x_t^*})$. Thus, the overall loss introduced by this triplet is given by

$$\mathcal{L}_{overall}(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-}) = \sum_{\ell=0}^{L} \alpha^{(\ell)}\mathcal{L}^{(\ell)}(\boldsymbol{x_t}, \boldsymbol{x_t^+}, \boldsymbol{x_t^-}) \tag{3.3}$$

Parameters $\Theta^{(\ell)}$, $\alpha^{(\ell)}$, and $W^{(\ell)}$ are learned during the online learning phase. The final optimization problem to solve in OAML at time t is therefore:

$$\begin{aligned} &\underset{\Theta^{(\ell)}, W^{(\ell)}, \alpha^{(\ell)}}{\text{minimize}} \quad \mathcal{L}_{overall} \\ &\text{subject to } ||f^{(\ell)}(\boldsymbol{x_t^*})||_2 = 1, \forall \ell = 0, \ldots, L. \end{aligned} \tag{3.4}$$

We evaluate the similarity and dissimilarity errors using an *adaptive-bound triplet loss* (ABTL) constraint (Gao et al., 2019) to estimate $\mathcal{L}^{(\ell)}$ and update $\Theta^{(\ell)}$, $W^{(\ell)}$ and $\alpha^{(\ell)}$.

Novel Class Detection. Novel classes may appear at any time in real-world monitoring streams (e.g., new attacks and new deceptions). To cope with such *concept-evolving* data streams, we include a deception-enhanced novel class detector that extends traditional classifiers with automatic detection of novel classes before the true labels of the novel class instances arrive.

Data stream classification. Novel class detection observes that data points belonging to a common class are closer to each other (*cohesion*), yet far from data points belonging to other classes (*separation*). Building upon ECSMiner (Masud et al., 2011; Al-Khateeb et al., 2016), our approach segments data streams into equal, fixed-sized *chunks*, each containing a set of monitoring traces, efficiently buffering chunks for online processing. When a buffer is examined for novel classes, the classification algorithm looks for strong cohesion among outliers in the buffer and large separation between outliers and training data. When strong cohesion and separation are found, the classifier declares a novel class.

Training & model update. A new classifier is trained on each chunk and added to a fixed-sized ensemble of M classifiers, leveraging audit and attack instances (traces). After each iteration, the set of $M+1$ classifiers are ranked based on their prediction accuracies on the latest data chunk, and only the first M classifiers remain in the ensemble. The ensemble is continuously updated following this strategy and thus modulates the most recent concept in the incoming data stream, alleviating adaptability issues associated with concept drift (Masud et al., 2011). Unlabeled instances are classified by majority vote of the ensemble's classifiers.

Classification model. Each classifier in the ensemble uses a k-NN classification, deriving its input features from Bi-Di and N-Gram feature set models. Rather than storing all data points of the training chunk in memory, which is prohibitively inefficient, we optimize space utilization and time performance by using a semi-supervised clustering technique based on Expectation Maximization (E-M) (Masud et al., 2008). This minimizes both intra-cluster dispersion and cluster impurity, and caches a summary of each cluster (centroid and frequencies of data points belonging to each class), discarding the raw data points.

Feature transformation. To make the learned representations robust to partial corruption of the input patterns and improve classification accuracy, abstract features are generated from the original feature space during training via a *stacked denoising autoencoder* (DAE) (Vincent

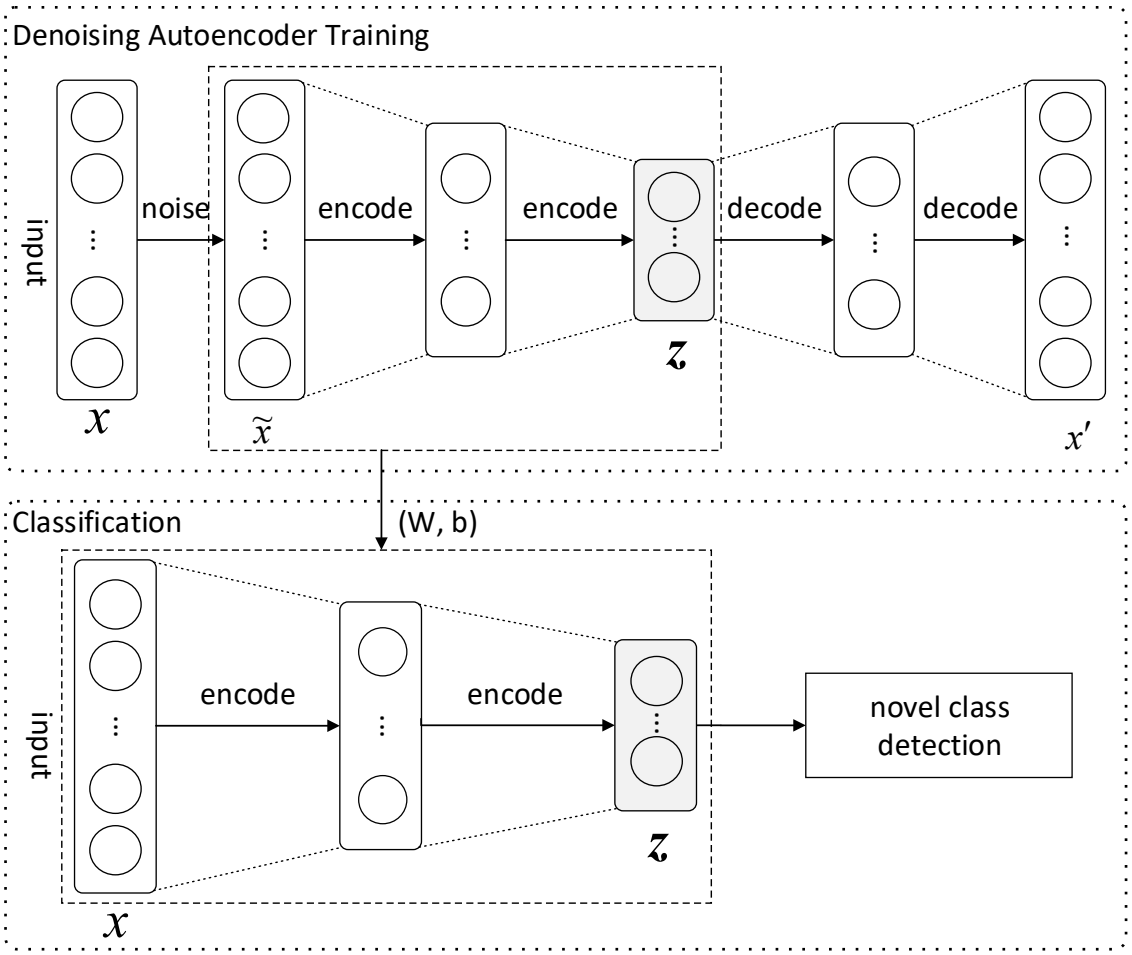


Figure 3.3: Overview of feature transformation

et al., 2008, 2010) using the instances of the first few chunks in the data stream. Stacked DAE builds a deep neural network that aims to capture the statistical dependencies between the inputs by reconstructing a clean input from a corrupted version of it, thus forcing the hidden layers to discover more robust features (yielding better generalization) and prevent the classifier from learning the identity (while preserving the information about the input).

Figure 3.3 illustrates our approach (Mustafa et al., 2017). The first step creates a corrupted version $\widetilde{x}$ of input $x \in \mathbb{R}^d$ using *additive Gaussian noise* (Chen et al., 2014). In other words, a random value v_k is added to each feature in x: $\widetilde{x}_k = x_k + v_k$ where $k = [1 \dots d]$ and $v_k \sim \mathcal{N}(0, \sigma^2)$ (cf., (Bengio, 2009)). The output of the training phase is a set of weights W and bias vectors b. We keep the learned weights and baises to transform the feature values of

the subsequent instances of the stream. After transforming the features of stream instances, these are fed back into our novel class detector for training.

One-class SVM Ensemble. Our approach builds an ensemble of one-class SVM classifiers. One-class SVM is an unsupervised learning method that learns the decision boundary of training instances and predicts whether an instance is inside it. We train one classifier for each class. For instance, if our training data consists of instances of k classes, our ensemble must contain k one-class SVM classifiers, each trained with one of the k class's instances.

During classification, once a new unlabeled instance x emerges, we classify it using all the one-class SVM classifiers in the ensemble.

We build our ensemble using the first few chunks of instances. During the classification of the stream, once novel class's instances emerge, we train a new one-class SVM classifier with the new novel class instances. Then we add the new classifier to the ensemble.

Attacker Evasion. To properly challenge deceptive defenses, it is essential to simulate adversaries who adapt and obfuscate their behaviors in response to observed responses to their attacks. Attackers employ various evasion techniques to bypass protections, including packet size padding, packet timing sequence morphing, and modifying data distributions to resemble legitimate traffic.

In our study, we considered three encrypted traffic evasion techniques published in the literature: *Pad-to-MTU* (Dyer et al., 2012), *Direct Target Sampling* (Wright et al., 2009), and *Traffic Morphing* (Wright et al., 2009). Pad-to-MTU (pMTU) adds extra bytes to each packet length until it reaches the Maximum Transmission Unit (1500 bytes in the TCP protocol). Direct Target Sampling (DTS) is a distribution-based technique that uses statistical random sampling from benign traffic followed by attack packet length padding. Traffic Morphing (TM) is similar to DTS but it uses a convex optimization methodology to minimize the overhead of padding. Each of these are represented using the traffic modeling

approach detailed in §3.1.1 and analyzed using the machine learning approaches detailed above.

3.2 Case Study

As a case study of our evaluation approach, we applied it to test DeepDig (Araujo et al., 2019), an IDS platform protecting deceptively honey-patched (Araujo et al., 2014) web servers. DeepDig is an anomaly-based IDS that improves its detection model over time by feeding attack traces that trigger honey-patch traps back into a classifier. This core feature makes it an advanced, intelligent defense that cannot be properly evaluated using static datasets.

3.2.1 Implementation

We implemented our evaluation framework atop 64-bit Linux. The data generation component is implemented using Python and Selenium (Selenium, 2019). The monitoring controller is 350 lines of node.js code, and leverages *tcpdump* (tcpdump, 2019), *editcap* (Linux Manual, 2019), and *sysdig* (Sysdig, 2019) for network and system call tracing and preprocessing. The machine learning modules are implemented in Python using 1200 lines of scikit-learn (Pedregosa et al., 2011) code for data preprocessing and feature generation. The novel class detection component comprises of about 250 lines of code to reference the Theano deep learning library (Theano Development Team, 2016) and ECSMiner (Masud et al., 2011). Finally, the OAML module was implemented with 500 lines of PyTorch (PyTorch, 2019) deep learning development framework code.

Model Parameters. In our experiments, SVM uses RBF kernel with Cost 1.3×10^5, and gamma is 1.9×10^{-6}. OAML employs a ReLU network with $n = 200$, $L = 1$, $k = 5$, learning rate of 0.3, lr decay of 1×10^{-4}, and ADAM optimizer. One-class SVM uses RBF kernel and Nu = 0.5. Novel class detection uses the DAE denoising autoencoder with $L = 2$, input

feature size = 6000, first layer = $\frac{2}{3}$ of input size, second layer = $\frac{1}{3}$ of input size, and additive Gaussian noise where $\sigma = 1.1$.

Noise injection. Rather than testing with existing, publicly available intrusion datasets (which are inappropriate evaluations of DeepDig, since they lack concept-relevance for deception and are generally stripped of raw packet data), our evaluation interleaves attack and normal traffic following prior work on defense-in-depth (Boggs et al., 2014), and injects benign payloads as data into attack packets to mimic evasive attack behavior. The generated traffic contains attack payloads against recent CVEs for which we created and tested realistic exploits, and our framework automatically extracts labeled features from the monitoring network and system traces to (re-)train the classifiers.

Dataset. Web traffic was generated from a separate host to avoid interference with the test bed server. To account for operational and environmental differences, our framework simulated different workload profiles (according to time of day), against various target configurations (including different background processes and server workloads), and network settings, such as TCP congestion controls. In total, we generated 42 GB of (uncompressed) network packets and system events over a period of three weeks. After feature extraction, the training data comprised 1800 normal instances and 1600 attack instances. Monitoring or testing data consisted of 3400 normal and attack instances gathered at unpatched web servers, where the distribution of normal and attack instances varies per experiment.

3.2.2 Experimental Results

Table 3.3 measures the accuracy of classifiers that were trained using deceptive servers, and then tested on attacks against *unpatched* servers. Attacks are uniformly distributed across all synthetic attack classes and variants described in §3.1.1. Each result is compared (in parentheses) against the same experiment performed without any deception. The results show that leveraging deception yields an 8–22% increase in classification accuracy, with an

Table 3.3: Detection rates (%) for scripted attack scenarios ($P_A \approx 1\%$) compared with results from non-deceptive training (parenthesized)

Classifier	***tpr***		***fpr***		***acc***		***F*$_2$**		***bdr***	
Bi-Di KNN	74.00	(-3.0)	0.01	(-41.2)	88.58	(+20.0)	78.00	(+19.4)	98.68	(+96.1)
N-Gram KNN	87.00	(+3.2)	0.01	(-5.1)	93.14	(+5.2)	89.00	(+1.3)	98.87	(+84.1)
Bi-Di DT	93.00	(+16.2)	0.01	(-41.2)	96	(+28)	94.00	(+35.0)	98.97	(+97.1)
N-Gram DT	12.00	(-62)	0.01	(-5.1)	58.58	(-30.0)	15.00	(-73.4)	92.68	(+78.0)
Bi-Di RF	92.00	(+15.0)	0.05	(-40.95)	93.58	(+24.04)	92.00	(+32.31)	94.68	(+92.13)
N-Gram RF	24.00	(-60.0)	0.01	(-5.1)	64.14	(-24.0)	28.00	(-60.0)	96.87	(+82.1)
Bi-Di OML	91.00	(+13.2)	0.01	(-41.2)	91.14	(+22.2)	90.00	(+30.3)	98.92	(+97.1)
N-Gram OML	65.00	(-19.9)	0.01	(-5.1)	88.58	(+0.0)	80.00	(-8.4)	98.50	(+83.8)
Bi-Di SVM	79.00	(+1.2)	0.78	(-40.5)	89.88	(+20.9)	78.69	(+19.0)	50.57	(+36.1)
N-Gram SVM	92.42	(+7.5)	0.01	(-5.1)	96.89	(+8.3)	93.84	(+5.5)	99.05	(+84.6)
Ens-SVM	93.63	(+8.8)	0.01	(-5.1)	97.00	(+8.4)	94.89	(+6.5)	99.06	(+84.6)

8–20% increase in true positives and a 5–41% reduction in false positives. Env-SVM achieves 97% accuracy with almost no false positives (0.01%).

These significant gains demonstrate that the detection models of each classifier learned from deception-enhanced data generalize beyond data collected in decoys. This showcases the classifier's ability to detect previously unseen attack variants. DeepDig thus enables administrators to add an additional level of protection to their entire network, including hosts that cannot be promptly patched, via the adoption of a honey-patching methodology.

Figure 3.4 shows that as the number of training attack classes (which are proportional to the number of vulnerabilities honey-patched) increases, a steep improvement in the true positive rate is observed, reaching an average above 93% for Ens-SVM, while average false positive rate in all experiments remains low ($< 1\%$). This demonstrates that deception has a *feature-enhancing* effect—the IDS learns from the prolonged adversarial interactions to detect more attacks.

Testing on an "unknown" vulnerability. We also measured our approach's ability to detect a *previously unseen*, unpatched remote code execution exploit (CVE-2017-5941) carrying attack payloads (classes 17–22) resembling the payloads that have been used to

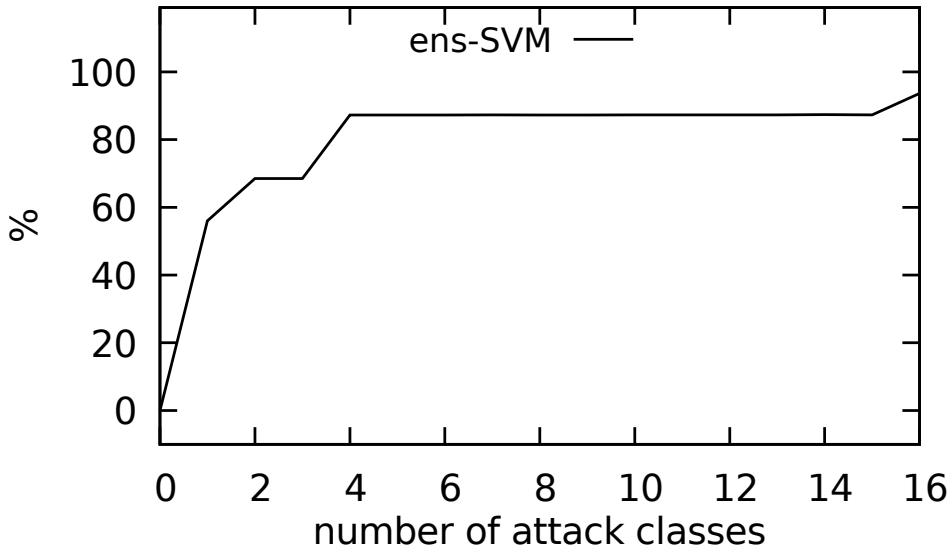


Figure 3.4: Ens-SVM classification *tpr* for 0–16 attack classes for training on decoy data and testing on unpatched server data.

exploit honey-patched vulnerabilities (CVE-2014-6271). In this experiment, CVE-2017-5941 is used as an n-day vulnerability for which no patch has been applied. The resulting 98.6–99.8% *tpr* and 0.01–0.67% *fpr* show that crook-sourcing helps the classifier learn attack patterns unavailable at initial deployment, but revealed by deceived adversaries during decoy interactions, to learn exploits for which the classifier was not pre-trained.

False alarms. Figure 3.5 plots the false positive rates for classifiers that have undergone 30 incremental training iterations, each with 1–30 normal/attack instances per class. With just a few attack instances (≈ 5 per attack class), the false positive rates drop to almost zero, demonstrating that DeepDig's continuous feeding back of attack samples into classifiers greatly reduces false alarms.

3.2.3 Base Detection Analysis

In this section we measure the success of DeepDig in detecting intrusions in the realistic scenario where attacks are a small fraction of the interactions. Although risk-level attribution for

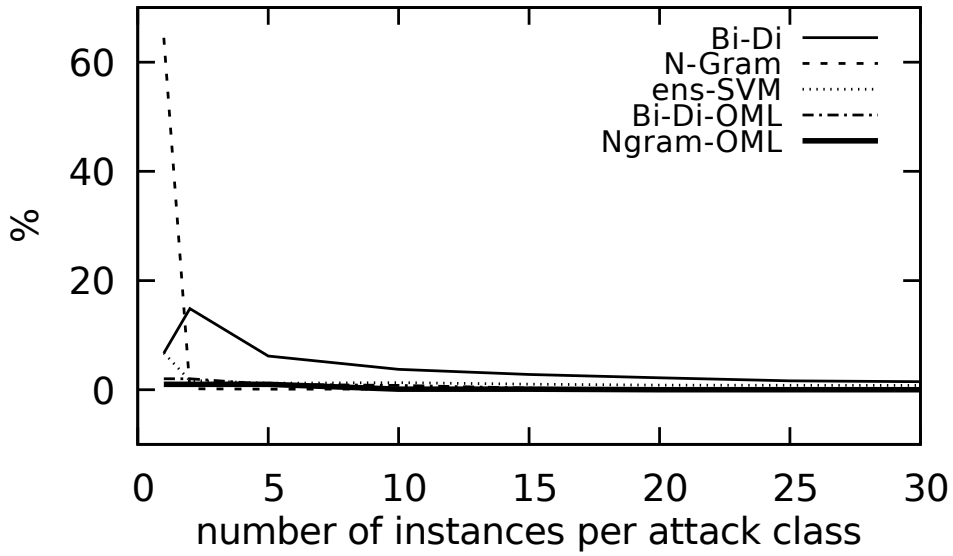


Figure 3.5: False positive rates for various training set sizes

cyber attacks is difficult to quantify in general, we use the results of a prior study (Dudorov et al., 2013) to approximate the probability of attack occurrence for the specific scenario of *targeted attacks against business and commercial organizations*. The study's model assumes a determined attacker leveraging one or more exploits of known vulnerabilities to penetrate a typical organization's internal network, and approximates the *prior* of a directed attack to $P_A = 1\%$ (based on real-world threat statistics).

To estimate the success of intrusion detection, we use a *base detection rate* (*bdr*) (Juarez et al., 2014), expressed using the Bayes theorem:

$$P(A|D) = \frac{P(A)\ P(D|A)}{P(A)\ P(D|A) + P(\neg A)\ P(D|\neg A)]}, \tag{3.5}$$

where A and D are random variables denoting the occurrence of a targeted attack and the detection of an attack by the classifier, respectively. We use *tpr* and *fpr* as approximations of $P(D|A)$ and $P(D|\neg A)$, respectively.

The final columns of Tables 3.3–**??** present the *bdr* for each classifier, assuming $P(A) = P_A$. The parenthesized comparisons show how our approach overcomes a significant practical

problem in intrusion detection research: Despite exhibiting high accuracy, typical IDSes are rendered ineffective when confronted with their extremely low base detection rates. This is in part due to their inability to eliminate false positives in operational contexts. In contrast, the *fpr*-reducing properties of deception-enhanced defense facilitate much more effective detection of intrusions in realistic settings, with *bdr* increases of up to 97%.

3.2.4 Monitoring Performance

To assess the performance overhead of DeepDig's monitoring capabilities, we used *ab* (Apache HTTP server benchmarking tool) to create a massive user workload (more than 5,000 requests in 10 threads) against two web server containers, one deployed with network and system call monitoring and another unmonitored.

Figure 3.6 shows the results, where web server response times are ordered ascendingly. Our measurements show average overheads of 0.2×, 0.4×, and 0.7× for the first 100, 250, and 500 requests, respectively, which is expected given the heavy workload profile imposed on the server. Since server computation accounts for only about 10% of overall web site response delay in practice (Souders, 2007), this corresponds to observable overheads of about 2%, 4%, and 7% (respectively).

While such overhead characterizes feasibility, it is irrelevant to deception because unpatched, patched, and honey-patched servers are all slowed equally by the monitoring activity. The overhead therefore does not reveal which apparent vulnerabilities in a given server instance are genuine patching lapses and which are deceptions, and it does not distinguish honey-patched servers from servers that are slowed by any number of other factors (e.g., fewer computational resources).

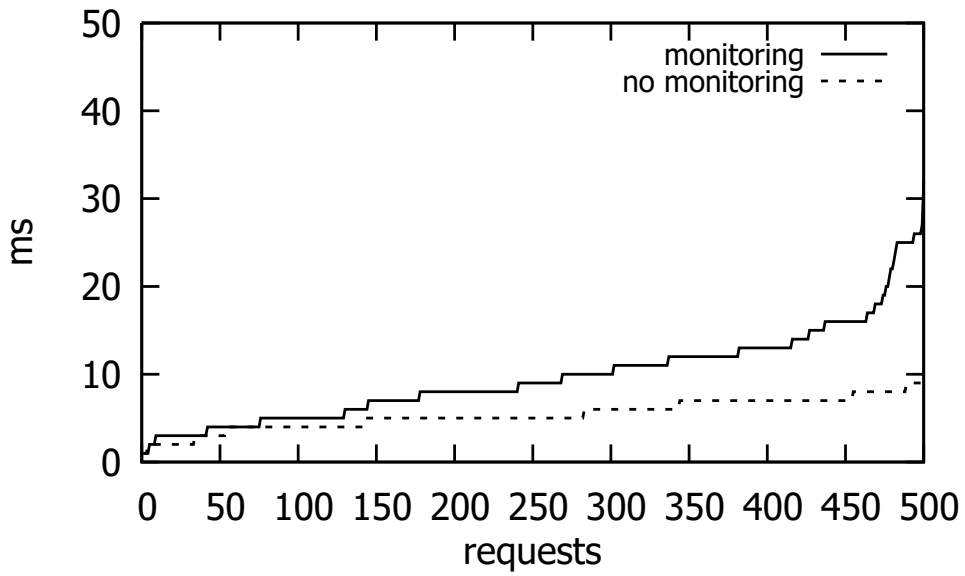


Figure 3.6: DeepDig performance overhead measured in average round-trip times (workload ≈ 500 req/s)

3.2.5 Resistance to Attack Evasion Techniques

Table 3.4 shows the results of the deceptive defense against our evasive attack techniques compared with results when no evasion is attempted. In each experiment, the classifier is trained and tested with 1800 normal instances and 1600 *morphed* attack instances.

Our evaluation shows that the tpr drops slightly and the fpr increases with the introduction of attacker evasion techniques. This shows that the system could resist some of the evasions but not all. However, we can conclude that an increase in the frequency of classifier retraining may be needed to accommodate the drop in performance. This may be a challenge as shorter time interval results in fewer data points to retrain the classifier to maintain their detection performance.

3.2.6 Novel Class Detection Accuracy

To test the ability of our novel class classifier to detect novel classes emerging in the monitoring stream, we split the input stream into equal-sized chunks. A chunk of 100 instances is

Table 3.4: Detection performance in adversarial settings

Evasion technique	***tpr***	***fpr***	***acc***	F_2
No evasion	**93.63**	**0.01**	**97.00**	**99.06**
pMTU	75.84	0.96	85.78	79.57
DTS	82.78	6.02	87.58	84.91
TM	79.29	6.17	85.52	81.91

classified at a time where one or more novel classes may appear along with existing classes. We measured the *tpr* (total incremental number of actual novel class instances classified as novel classes) and the *fpr* (total number of existing class instances misclassified as belonging to a novel class).

Table 3.5 shows the results for OneSVM and ECSMiner. Here ECSMiner outperforms OneSVM in all measures. For example, for Bi-Di features, ECSMiner observes an *fpr* of 26.66% while OneSVM reports an *fpr* of 31.88%, showing that the binary-class nature of ECSMiner is capable of modeling the decision boundary better than OneSVM. To achieve better accuracy, we augmented ECSMiner with extracted deep abstract features using our stacked denoising autoencoder approach (DAE & ECSMiner). For DAE, we used two hidden layers (where the number of units in the first hidden layer is 2/3 of the original features, and the number of units in the second hidden layer is 1/3 of the first hidden layer units). For the additive Gaussian noise, which is used for data corruption, we assigned $\sigma = 1.1$. As a result, *fpr* reduced to a minimum (0.01%), showing a substantial improvement over ECSMiner. Notice that using the abstract features with OneSVM does not help as shown in the table.

While effective in detecting concept drifts, our novel class detection technique requires a (semi-)manual labeling of novel class instances. In our future work, we plan to investigate how to automatically assign labels (e.g., deceptive vs. non-deceptive defense response) to previously unseen classes.

Table 3.5: Novel attack class detection performance

Features	**Classifier**	***tpr***	***fpr***
Bi-Di	OneSVM	44.06	31.88
	DAE & OneSVM	76.54	85.61
	ECSMiner	74.91	26.66
	DAE & ECSMiner	**84.73**	**0.01**
N-Gram	OneSVM	54.25	45.13
	DAE & OneSVM	80.09	71.49
	ECSMiner	76.36	34.89
	DAE & ECSMiner	**89.67**	**2.95**

3.3 Related Work

Deception-enhanced IDS. Our evaluation methodology is designed to assess adaptive, deception-enhanced IDS systems protecting web services. Examples from the literature include shadow honeypots (Anagnostakis et al., 2005, 2010), Argos (Portokalidis et al., 2006), Honeycomb (Kreibichi and Crowcroft, 2004), and DAW (Tang and Chen, 2005).

Synthetic Attack Generation. Our approach was inspired by WindTunnel (Boggs et al., 2014), which is a synthetic data generation framework for evaluating (non-deceptive) security controls. WindTunnel acquires data from network, system call, file access, and database queries and evaluates which of the data sources provides better signal for detection remote attacks. The DETER (Benzel et al., 2006) testbed provides a framework for designing repeatable experiments for evaluating security of computer systems.

CHAPTER 4

MITIGATING CYBERATTACKS USING DOMAIN ADAPTATION TECHNIQUE [1]

4.1 Introduction

In recent years, a large number of organizations have encountered sophisticated network attacks, including advanced persistent threats (APT) (Fireeye, a). For example, Figure 4.1 displays a snapshot of a trace file showing scanning session of an attacker using the *nmap* tool to scan a victim's network. Our goal in this work is to link the low level event traces to the higher concept of actions called tactics deployed by an attacker. Bridging this gap is challenging for multiple reasons. First, extracting useful features to train the classifier is challenging. For example, a typical event trace consists of noisy operating system events that do not correspond to attacker actions.

Additionally, the scarcity of labeled data has limited the use of machine learning techniques in detecting APT attacks. However, some domains tend to suffer more attacks and the data collected can be used to train classifiers in another domain. For example, some operating systems encounter more attacks and therefore attract more attacker actions. The data collected from these attacks can be used to detect attacks in platforms with less available labeled data. However, there is the challenge of adapting data from different domains to train a machine learning model. For example, data collected from Linux machines cannot be directly used to predict attack events on a host deploying the Windows operating system. That is, the *open* system call in Linux is named *OpenFile* system call in Windows operating

[1]This chapter contains material previously published as: Li, Yifan., Yang. Gao, Gbadebo Ayoade, Hemeng Tao, Latifur Khan, and Bhavani Thuraisingham. "Multistream classification for cyber threat data with heterogeneous feature space." In *The World Wide Web Conference, WWW '19*, New York, NY, USA, pp. 2992–2998. ACM. Ayoade led the cyberthreat detection aspect of this research

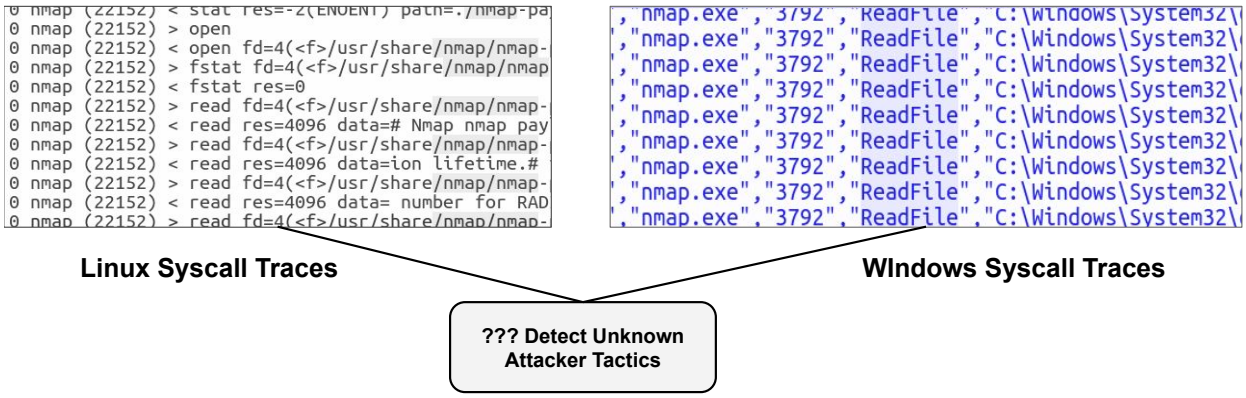


Figure 4.1: Sample Trace File showing syscall traces for a attacker scanning session with nmap network scanning tool

Table 4.1: Windows vs Linux system call mismatch

Operation	**Syscall in Linux Sysdig**	**Syscall Windows by sysmon**
Read file	read	ReadFile
Write to file	Write	WriteFile
Open file	Open	OpenFile
Close file	Close	CloseFile
Socket accept	Accept	TCP Accept
Socket connect	Connect	TPC Connect
Flush registry	-	RegFlushKey

system resulting in a string mismatch between these two similar events as shown in Table 4.1.

To overcome this limitation, a naïve approach would be to manually map the feature space of the domains to each other. However, this process is cumbersome and slow. We overcome this challenge by proposing a novel domain adaptation method (Li et al., 2019) that automatically learns a new feature space that projects the individual feature spaces across multiple domains to a latent space.

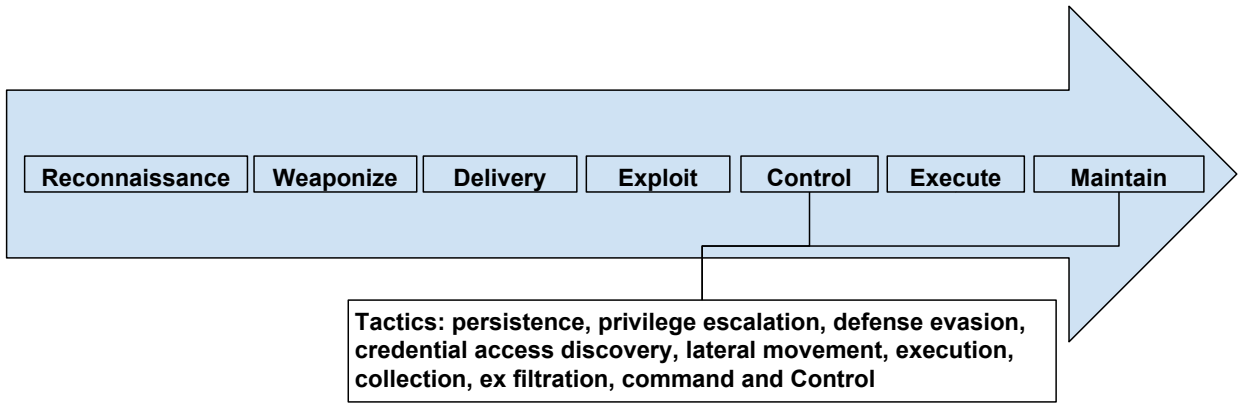


Figure 4.2: MITRE ATT&CK Kill Chain Phase

4.2 Background

4.2.1 APT attacks

In this section, first, we present relevant background information on the MITRE ATT&CK framework. Second, we discuss the domain adaptation technique.

4.2.2 MITRE ATT&CK/Mandiant Kill Chain Phase, Tactics and Techniques

ATT&CK is a threat categorization framework developed by MITRE to identify unique attacker behaviors and actions. They categorize the attacks into tactics and techniques. (MITRE, MITRE). Attackers usually perform their attacks in phases as shown in Figure 4.2. Attackers can methodically complete their attack mission without being detected by defenders. Knowledge of these attack tactics and techniques can help defenders in detecting and mitigating these attacks. These phases include ***reconnaissance, weaponization, delivery, exploitation, command and control, and execution and maintenance***. The early phase of the attack life cycle consists of reconnaissance, weaponization, delivery, exploitation while the late phase consists of command and control, execute and maintain.

4.2.3 DarkNet

In recent years, there has been an increase in data theft which is usually listed on illegal websites that are often hidden from the open internet. This hidden websites are usually anonymous and are usually hosted on the Tor network. As a result, these websites are called darknet. Because of the nature of these websites, illegal activities such as the sale of stolen credit-card numbers, passwords, social security numbers, email addresses. Other dangerous items include malicious executable, viruses, ransomware, cryptoware are usually listed on these websites. For example, Figure 4.3 shows a real-world listing of a posting for a request to buy a "crypter" for a ransomware attack. Likewise, the second post shows a request for the sale of stolen UK email addresses.

We aim to create a framework that enables us to extract what words in the listings are products. With such extraction, we can create an automated way to scan darknet for illegal goods sold which can then be utilized by law enforcement agencies or even credit card companies to track stolen data.

To achieve this goal, we will require a lot of labeled data especially, across multiple darknet domains. By leveraging domain adaptation, we can use data labeled from one darknet domain on another data collected from a different target domain.

4.3 Proposed Approach

4.3.1 Domain Adaptation Approach

The goal of Multisource Domain Adaptation (MSDA) is to learn a feature space that is common among two distinct data set. Figure 4.4 illustrates this idea where the source data is a two-dimensional and the target data is a one-dimensional data. With MSDA, we can learn a latent feature space where the original and the latent space features are from the same distribution. MSDA also preserves the structure of the data after transformation, that is, the

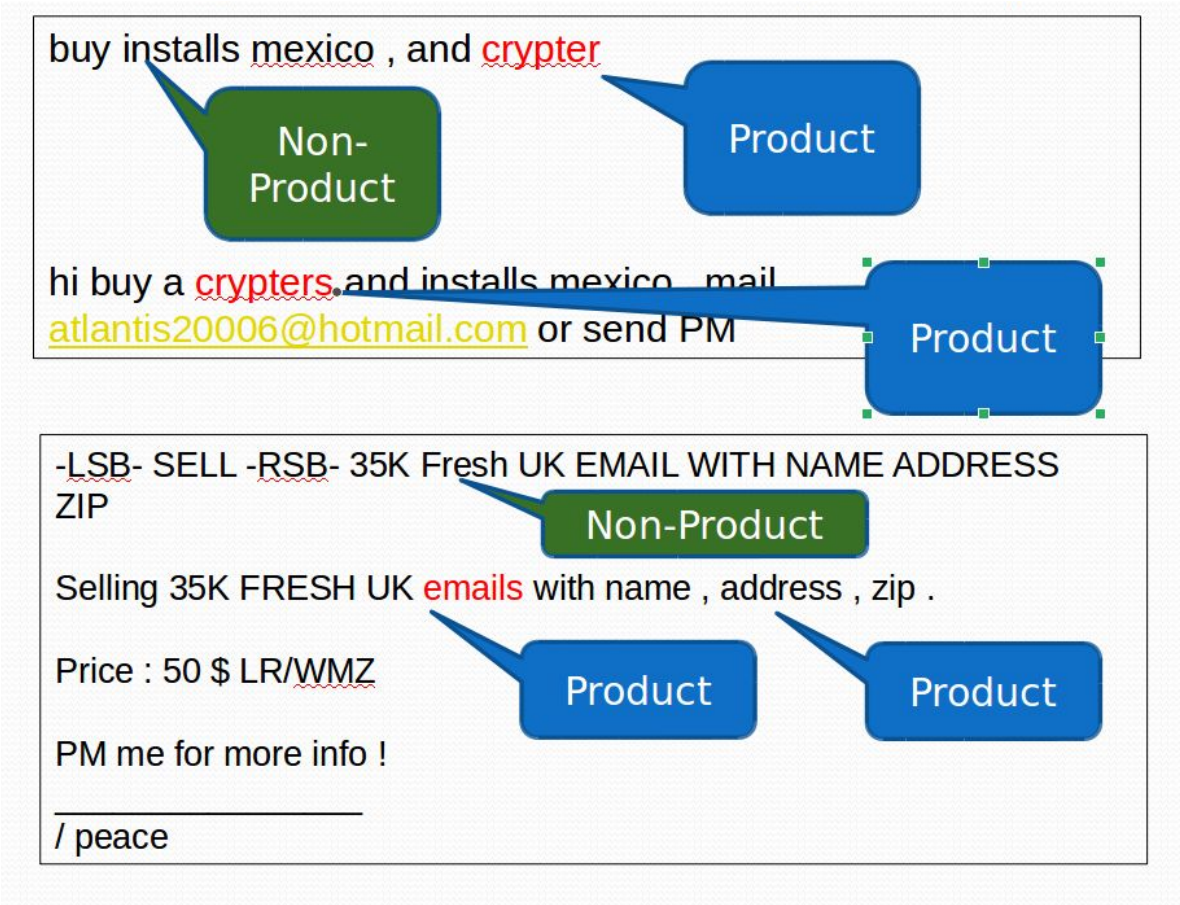


Figure 4.3: Dark net selling and buying post

decision boundaries of the different classes is preserved. The aim is to learn a feature space where the measure of similarity between the original and latent space feature is maximized.

4.4 Feature Extraction

4.4.1 Packet Features Analysis

For feature extraction, we extract both Uni-burst and Bi-Burst features as shown in Table 4.2. A Burst is a sequence of consecutive packets that are transmitted in the same direction. We use only the header information of the packets. In our case, we use all packets transmitted which include both TCP and UDP packets.

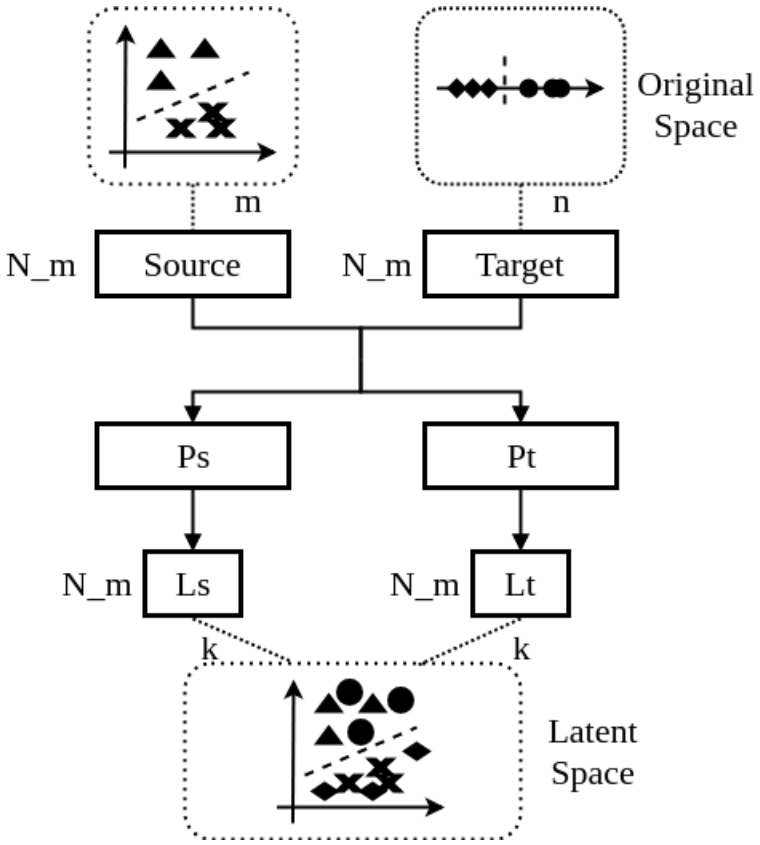


Figure 4.4: Feature Space Projection

Require: Labeled source data S, Unlabeled target data T,
Similarity parameter β.
Ensure: Labels predicted on T.
1: /* Initialization */
2: $B_s, B_t \leftarrow readData(S, T)$
3: /* DA for initial buffer */
4: $W_s, W_t \leftarrow genProjectionFunction(B_s, B_t, \beta)$
5: $L_s, L_t \leftarrow genProjectionMatrix(B_s, B_t, W_s, W_t)$
6: $M \leftarrow buildModel(L_s, Y_s)$
7: /* Generate predictions */
8: $\hat{y}_t \leftarrow getPrediction(M, L_t)$

Uni-burst features A Uni-burst consists of the burst size, time and count. The burst size is the summation of all the individual packet payload lengths in the burst. The time is the

Table 4.2: Packet, uni-burst, and bi-burst features

Category	Features
Packets	Packet Length
Uni-Burst	Uni-Burst Size,Uni-Burst Count, Uni-Burst Time
Bi-Burst	Bi-Burst Size, Bi-Burst Time

total transmission time of the the burst and the count is the total number of packets in the burst.

Bi-burst features For Bi-burst, we use the Bi-burst size and time. For each consecutive burst, we take the burst size, time for the downlink and uplink burst and get the *Tx-Rx-bursts* and *Rx-Tx-bursts*. The Bi-burst feature helps to capture dependencies within the packet flow on the network.

4.4.2 System Call Analysis

We collected host machine traces which includes a collection of operating system events, where each event consist of event types like (e.g., *open*, *read*, *select*), process name and process arguments . Our prototype implementation was developed for Linux x86_64 systems and Windows systems. We build histograms from these system calls using N-Gram. With N-Gram, we extract features from *contiguous sequences of* system call*s*.

There are four feature types: *Uni-events* are system calls with single sequence. *Bi-events* are consist of a sequence of two consecutive events as a single feature.

Likewise, *tri-* and *quad-events* consist of sequence of three and four consecutive events.

4.4.3 Feature Extraction for Darknet dataset

In order to extract features for classifying which words are product or not, we leverage the Penn Part of Speech Tags (POS) and Stanford postagger system. With the POS tagger, we extract features such as if a word is a noun, pronoun, verb or adjective. We also look at the

position of the word in the post. We use one-hot encoding to encode which part of speech tag a word belongs to by setting the corresponding vector position as one or zero.

4.5 Domain Adaptation

By leveraging domain adaptation, we propose a module that maps both source and target data to an optimized latent subspace. First, we learn a projection function for the source and target data instances. The learned function is then used to transform target or testing instances to a latent feature space representation. Second, we train a classifier based on the latent features and the classifier is used to classify target data instances. subsection 4.5.1 provides more details on these steps.

4.5.1 Training and Domain Adaptation

For our framework, the source and target data is stored in B_s and B_t respectively. Based on our problem definition, we know that data contained in B_s have true labels, and data in B_t do not have labels. We formulate the domain adaption method as an optimization problem (Wei et al., 2016). For our approach, we ensure that our method preserves the structure of the data across the latent feature space. The preservation is achieved by using a proper co-regularizer in the third term of Equation 4.1.

We now describe how it works.

First, we load the source and target data into the memory, it results in the following data: source data matrix $B_s \in \mathbb{R}^{N_m \times m}$; source labels vector $Y_s \in \mathbb{R}^{N_m \times 1}$, and; target data matrix $B_t \in \mathbb{R}^{N_m \times n}$. Where N_m is the number of data in the dataset. Therefore, we minimize the following objective function to learn the best projection strategy of L_s and L_t in the latent feature space:

$$\mathcal{O} = \min_{L_s, L_t} \ell\left(B_s, L_s\right) + \ell\left(B_t, L_t\right) + \beta \mathbf{D}\left(L_s, L_t\right) \tag{4.1}$$

where B_s, B_t are source and target data in memory; the projected data is represented by L_s, L_t; $\ell(\cdot,\cdot)$ represent the distance metric between the projected and the original data instances. We represent the co-regularizer with $\mathbf{D}(\cdot,\cdot)$ which preserves the projected domain similarities between L_s and L_t. β is a used to define how similar the projected data should be. The initial two terms are used to preserve the structural form of the original data. Therefore, the matrix trace norm is a Frobenius norm of the loss function $\ell(\cdot,\cdot)$ which is defined as $\ell(B_s, L_s)$ which is equal to $\|B_s - L_s W_s\|_F^2$, and $\ell(B_t, L_t)$ equals $\|B_t - L_t W_t\|_F^2$, where W_s and W_t are projection functions. Note that here we are not applying the alternative definition such as $\ell(B_s, L_s)$ as $\|B_s W_s - L_s\|_F^2$, since this definition will always lead to a trivial solution $W_s = 0$ and $L_s = 0$, thus $B_s W_s = L_s = 0$ will always minimize the objective function. From Equation 4.1 the projected data should preserve original structure of the data.

In addition, $\mathbf{D}(Ls, L_t)$ is defined as follows:

$$\mathbf{D}(L_s, L_t) = \frac{1}{2}\ell(B_s, L_t) + \frac{1}{2}\ell(B_t, L_s) \tag{4.2}$$

Equation 4.2 defines the cross-similarity mean value between the projected and the original instances. Finally, we minimize the the differences between $\mathbf{D}(L_s, L_t)$ by using the parameter β to control similarity co-regularization.

We obtain the following objective function as follows:

$$\begin{aligned}\mathcal{O} &= \|B_s - L_s W_s\|_F^2 + \|B_t - L_t W_t\|_F^2 \\ &+ \frac{1}{2}\beta\|B_s - L_t W_s\|_F^2 + \frac{1}{2}\beta\|B_t - L_s W_t\|_F^2\end{aligned} \tag{4.3}$$

We therefore adopt an alternative formula to solve this problem by iteratively fixing one of the projection matrices until the remaining one converges. That is, we can take the derivative of $\mathcal{O}$ with regard to W_s and W_t. Under this conditions, our formula is defined as follows accordingly.

Table 4.3: Datasets Summary

Data type	Datasets	# features	# instances
APT detection	Win	28	10587
	Linux	105	10801
Dark web	BlackHat	38	32414
	Nulled	38	9930
	Darkode	67	100,000
	Hack	67	100,000

$$\begin{aligned} \frac{\partial \mathcal{O}}{\partial W_s} &= (2+\beta)W_s - 2L_s^\top B_s - \beta L_t^\top B_t \\ \frac{\partial \mathcal{O}}{\partial W_t} &= (2+\beta)W_t - 2L_t^\top B_t - \beta L_s^\top B_t \end{aligned} \tag{4.4}$$

Combining Equation 4.1 and 4.2 together, then we can take the derivative of $\mathcal{O}$ with regard to W_s and W_t re respectively. According to Long et al. (Long et al., 2008), setting both partial derivatives to zero would generate the optimal solution based on KTT conditions. Consequently, the projection function for both source and target data can be formulated as:

$$\begin{aligned} W_s &= \frac{\beta}{2+\beta} L_t^\top B_s + \frac{2}{2+\beta} L_s^\top B_s \\ W_t &= \frac{\beta}{2+\beta} L_s^\top B_t + \frac{2}{2+\beta} L_t^\top B_t \end{aligned} \tag{4.5}$$

After performing the steps derived above, we can obtain the optimal projection of the data instances to the latent space.

4.6 Domain Adaptation Evaluation

4.6.1 Dataset

Win and **Linux** Table 4.3 gives a summary of our dataset. We collect both system-call events from windows host machines and Linux host machines. Each event consists of event

types like (e.g., open, read, select), process name and process arguments. Our generated dataset consists of various attack scenarios which include malicious exlfiltration of data, password theft, setting up a malicious port listening service for command and control, and using **nmap** to scan the victim's enterprise system. The benign instances were generated from web browsing activities and ssh login sessions.

Using sysdig system trace tool, we collected traces of system call events for Linux and for windows, we utilized procmon tool. In total, we collected 10,587 instances for Windows and 10,801 instances for Linux. In total, we collected 10,587 and 10,801 trace instances. We build histograms from these system calls using N-Gram. We extracted 28 unique features for Windows system and 105 features for Linux systems. To evaluate our approach, we generated the APT dataset using the APT simulator tool (NextronSystems, NextronSystems). This tool allows us to generate attacks based on the MITRE attack tactics framework. This tool uses real-world attack toolkits found in the wild from different APT attack campaigns, therefore allowing us to evaluate our system with realistic attack payloads.

For each tactic, we executed the script and collected the network and system call traces over different system execution windows using `tcpdump` and `procmon` respectively.

BlackHat, Nulled, Darkode and **Hack** (Portnoff et al., 2017) Figure 4.3 are datasets collected from dark website forums where illegal goods such as stolen passwords, credit card numbers are listed and sold. We only evaluated our approach on forums listed in English. The Hack forum contains a mixture of cyber-security and computer gaming blackhat and noncybercrime products; Nulled forum contains data stealing tools and services; Darkode focused on cybercriminal wares, including exploit kits, spam services, ransomware programs, and stealthy botnets; BlackHat focuses on blackhat search engine optimization (SEO) techniques. The forum started in October 2005 and is still active, although it has changed in character over the past decade. For this dataset, we want to obtain which products are listed on the dark web forums. Since the forums contain sentences like, "*I want to sell ransomware*

Table 4.4: Comparison of performance (Error %)

Data type	Dataset	OTL	HeMap-S	HeMap-L	MSDA-S	MSDA-L
Cyber security	Win → Linux	28.53 ± 0.88	30.55 ± 0.86	29.08 ± 0.58	23.73 ± 0.62	**21.33 ± 1.62**
	BlackHat → Darkode	22.42 ± 0.85	24.70 ± 1.00	25.36 ± 1.00	22.12 ± 1.29	**20.82 ± 0.64**
	Nulled → Hack	28.86 ± 1.54	26.78 ± 0.95	29.26 ± 0.50	25.47 ± 1.06	**24.75 ± 1.36**

tool for $50". We want to extract which of the words in this sentence are products. To accomplish this, we extracted features for each word and labeled it as a product or not. We use Penn Part of Speech Tags (Santorini, 1990) to extract features Nulled and BlackHat dataset and the Stanford pos- tagger system (De Marneffe and Manning, 2008) for the remaining dataset.

4.6.2 Results

Table 4.4 shows a summary of the average prediction error on the target data stream given as T: $\frac{A_{wrong}}{m}$, where A_{wrong}, where A_{wrong} represents the number of incorrectly classified instances, and m represents the total number of target data instances. For our experiment, we trained on Windows data and tested on Linux data. For this, we obtained an error rate of 21.33% for MSDA-L compared to 28.53% for OTL, 30.55 for HeMap-S. Similarly, our method performed better than other methods for the darknet dataset. For this, we obtained an error rate of 20.82% for MSDA-L compared to 25% for HeMap-L.

4.7 Related Work

In this section, we discuss the related work. First, we discuss previous approaches in intrusion detection systems based on machine learning. Second, we discuss related work in domain adaptation methods. Lastly, we discuss previous work on threat attack simulation to help defenders evaluate their defense strategy.

Domain Adaptation Several methods have been proposed to learn a common feature representation for domain adaptation (Ben-David et al., 2010; Pan and Yang, 2010). In

addition, Pan et al. (Pan et al., 2008) uses a dimension reduction method, Maximum Mean Discrepancy Embedding (MMDE) by minimizing the projection of the difference in distributions of data to a subspace. (Shi et al., 2010) projects a latent feature space for both source and target domains by learning a linear objective function. In our work, we developed a novel domain adaptation technique that leverages a new objective function to learn the feature projection across heterogeneous domains.

Our work improves existing frameworks by constructing and introducing a co-regularizer in the objective function, so that structure of data from both source and target data sets can be preserved.

Advanced Persistent Attack Threat Emulation (NextronSystems, NextronSystems) built a shell-based system that simulates advanced persistent attacks based on the MITRE ATT&CK framework. (Applebaum et al., 2016, 2017) developed a similar system with a web-based front end for more complex simulation of various combinations of attacker actions. These systems allow defenders to simulate real attacker scenarios and evaluate their detection systems against such attackers.

CHAPTER 5

AUTOMATED THREAT REPORT CLASSIFICATION OVER MULTI-SOURCE DATA [1]

5.1 Introduction

In recent years, a large number of organizations have encountered sophisticated network attacks, including advanced persistent threats (APT) (Fireeye, a). Most APT attacks use techniques that easily evade generalized off-the-shelf defense mechanisms. A typical APT attacker may use normal non-malicious software programs to complete an attack. For example, programs deployed may include Power-shell, which is popular for performing remote command execution on Windows, or remote desktop protocol program or FTP to propagate themselves within the victims network without detection.

With the goal of increasing security by creating awareness among concerned individuals regarding such attacks, these organizations periodically share threat information in the form of reports. To create a centralized repository of attack information about adversarial behavior, MITRE (MITRE, MITRE; Hutchins et al., 2011) has proposed a framework called ATT&CK (Adversarial Tactics, Techniques and Common Knowledge) for categorizing attacker actions into *kill-chain phase*, *tactics* and *techniques*. Here, the kill-chain phase describes the current phase in the attack strategy deployed by an attacker. Tactics provide course-grained information representing the high level intent of why an attacker exhibits a particular behavior on the victim's network or system infrastructure, and techniques provide fine-grained information representing how an attacker completes an attack activity.

[1] This chapter contains material previously published as: Gbadebo Ayoade, Swarup Chandra, Latifur Khan, Kevin Hamlen, and Bhavani Thuraisingham. "Automated Threat Report Classification over Multi-source Data." *In Proceedings of the 4th IEEE International Conference on Collaboration and Internet Computing (CIC)*, pp. 236–245, October 2018. Ayoade led the machine learning aspect including the design, implementation and evaluation of the approaches used in this research.

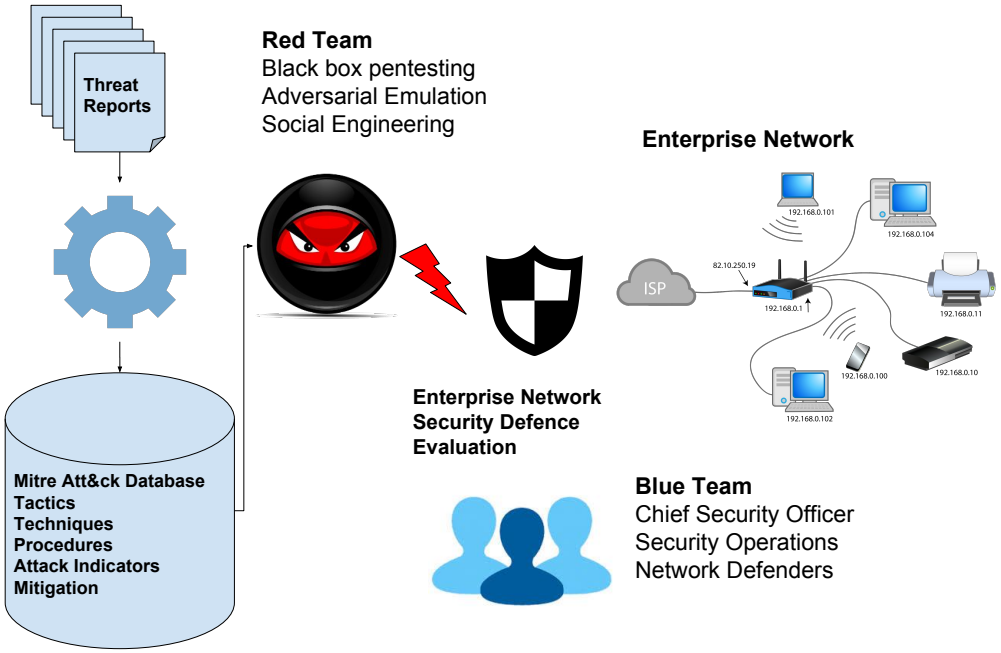


Figure 5.1: Adversarial Emulation and Enterprise System Defense Evaluation using MITRE Att&ck Collaborative Framework

As shown in Figure 5.1, MITRE collects different APT reports from different organizations which are then manually examined to populate the MITRE ATT&CK repository for tactics and techniques deployed by APT attackers. By leveraging the MITRE ATT&CK centralized repository, red teams which may include penetration testing teams and external system security auditors can emulate attacker behaviors and provide a more comprehensive attack coverage which can then be used to determine the strength of the defense teams usually called the blue teams. Blue teams can leverage the mitigation available in this repository to evaluate their defense strategy. For example, a red team can consult the MITRE ATT&CK framework to determine the tactics and techniques that may be used to complete a lateral movement attack on a victim's network and emulate it. The blue team is expected to detect this attack by deploying its security defense infrastructure. Our work extends the MITRE ATT&CK with a collaborative framework that provides a system to automatically extract

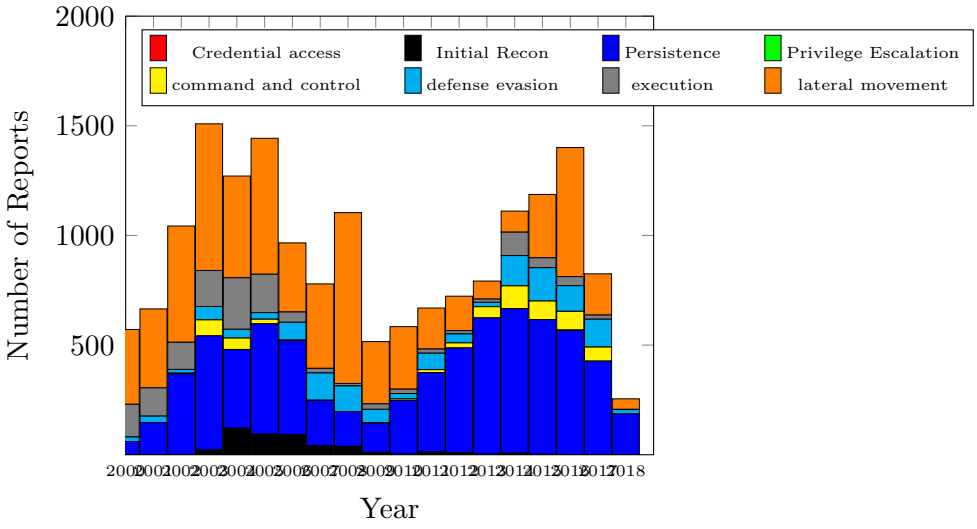


Figure 5.2: Threat Report Data Generation Distribution between Year 2000 and 2018

and categorize different threat reports from different organizations that can be leveraged to update the central repository.

With an increasing trend in the number of threat reports generated from various organizations as shown in Figure 5.2 (Fireeye, b), automated parsing of threat reports is necessary to facilitate effective usage. From the figure, the average number of reports generated per year is around 1,000 documents per year. Processing these reports manually to extract information is inefficient and cumbersome since the reports are not usually generated with evenly distributed frequency throughout the year as some days may have more reports than others. Furthermore, these reports are generated by different organizations and employ different reporting formats. In addition, most threat reports are generated in an unstructured text format as shown in Figure 5.4 and do not follow a commonly standardized categorization of the attack behavior patterns. This results in the manual categorization of attacks, yielding to an erroneous and cumbersome process. (Ghaith et al., 2017). To facilitate these automated threat intelligence sharing, different report exchange formats such as STIX (STIX, STIX), TAXII (TAXII, TAXII), CVEs, and CWE have been proposed. Unfortunately, these for-

mats are not followed by different organizations resulting in difficulty to update the MITRE ATT&CK's central repository.

In this chapter (Ayoade et al., 2018), our aim is to help reduce search time for an analyst who wishes to reproduce the attack type for performing defense evaluation. We address the above challenges by building a machine learning classifier for automated threat report categorization that can generalize across reports from various organizations. We evaluate our approach as a multi-level classification problem. For this approach, we classify the threat reports first into tactics and then into techniques. In particular, we aim to leverage the comprehensive description of tactics and techniques provided in the MITRE ATT&CK framework for relevant information extraction to train a classifier. Unfortunately, due to the limited adoption of this framework in threat reports generated from various organizations, the availability of sufficient labeled data is limited. Reports with ATT&CK tactics and techniques are fewer than the overall reports, with only a few organizations following such a format. This creates bias in training data, which affects the performance of a classifier trained naïvely on the data. We address this limitation by applying a bias correction mechanism (Jiang and Zhai, 2007) which evaluates importance weights during classifier training, along with a confidence propagation strategy, for superior course-grained and fine-grained categorization. We empirically demonstrate the effectiveness of our approach on a large set of threat reports and show performance improvements over existing method.

The rest of the chapter is organized as follows. §5.2 provides the overview of our approach. §5.3 discusses the related work. §5.4 provides background on the MITRE ATT&CK framework and Bias correction methods. §5.5 provides the architecture of our system. §5.6 provides the evaluation of our approaches. . Finally, §7.3 concludes and discusses future work.

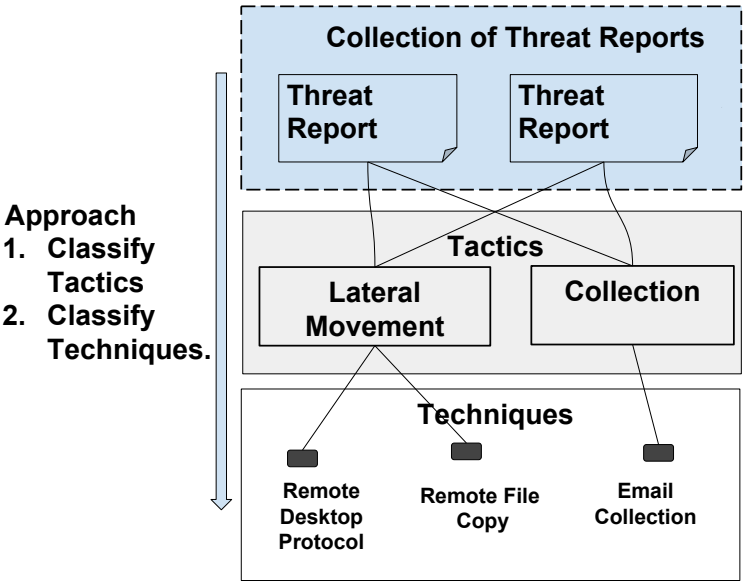


Figure 5.3: Tactics and Techniques classification of Threat Reports

5.2 Overview

Figure 5.3 shows the overview of our problem statement. Given a set of threat report documents, we want to determine which tactics and techniques are present in the report. For our approach, first we classify the tactics and secondly, we classify to the techniques used. Tactics give a high-level intent on why an attacker is performing an attack and the technique describes how an attacker completed the attack. In section 5.4, we will provide detailed explanation of tactics and techniques as given by the MITRE ATT&CK framework.

5.3 Related Work

Various industry efforts have been carried out to provide threat sharing formats that can be applied by security professionals to share threat data. They include the Open Indicator of Compromise format (`openIOC`) (OpenIOC, OpenIOC), Structured Threat Information eXpression (`STIX`) (STIX, STIX), Trusted Automated Exchange of Intelligence Information

(TAXII) (TAXII, TAXII), CVEs, and CWE. These formats leverage a machine-readable formats to exchange threat indicators like the skill level of attackers involved in an attack, the tools used, the attack phases, and the attack tactics used. MITRE ATT&CK covers why and how attackers perform advanced persistent attacks. Attackers use a variety of approaches to achieve their end goal including deploying CVE vulnerabilities.

In recent times, researchers have focused on techniques that automatically extract useful threat information from data available online from blogs and threat report websites. Hutchins et al. (Hutchins et al., 2011) provided a technique to categorize advanced persistent threat attacks to kill chain phases as described in the introduction. By classifying the attacker's actions into phases, defenders can comprehend attacker steps and seek to understand the attacker's motives.

A similar work, TTPDrill to this chapter by Ghaith et al. (Ghaith et al., 2017) applied NLP technique to extract threat actors, threat indicators and generate STIX standard formatted reports from unstructured data. First, our work differs from their work because TTPDrill used a simple lazy classification technique based on calculating similarity scores between two documents. Second, we applied a bias correction technique to address the challenge of sampling bias due to limited labeled training data, therefore, TTPDrill performs very poorly when the data sources for training and testing are very different. Third, we performed experiments on 2 datasets as compared to only one dataset by TTPDrill. In particular, TTPDrill focuses on extracting threat indicators from documents with short sentences of less than 900 words, while we focus on retrieving large and contextual documents containing more than 10,000 words for a set of tactics as shown in Table 5.3, where TTPDrill typically under-performs. In addition, our dataset comprises of more complicated documents with larger file size and more complex sentence structures. The individual file size for the TTPDrill is 4KB while the average files size for the second dataset we used is 15KB. Furthermore, our approach outperforms TTPDrill significantly as shown in the evaluation section §5.6.

In the Internal Reconnaissance stage, the intruder collects information about the victim environment. Like most APT (and non-APT) intruders, APT1 primarily uses built-in operating system commands to explore a compromised system and its networked environment.

Figure 5.4: Sample extract from data description for MITRE ATT&CK discovery tactic.

Xiaojing et al. (Liao et al., 2016) applied NLP techniques to automatically extract indicators of compromise such as botnet IPs, malware names from unstructured text to more standardized format. Our work differs from this work because we consider classifying threat reports into tactics and techniques an attacker employed to complete an attack. Our work focuses on the overall attacker behavior rather than just extracting tools used by the attacker. Burger et al. (Burger et al., 2014) classified various threat sharing technologies on how they inter-operate. By considering the different uses of cases of the various threat sharing technologies, they propose a way to unify these techniques for wider usage and adoption by security professionals.

5.4 Background

In this section, first, we will introduce the MITRE ATT&CK framework since our work leverages this framework. Next, we discuss existing bias correction methods to overcome bias sampling that may arise due to our dataset.

5.4.1 MITRE ATT&CK/Mandiant Kill Chain Phase, Tactics and Techniques

ATT&CK is a threat categorization framework developed by MITRE to identify unique attacker behaviors and actions. They categorize the attacks into tactics and techniques. (MITRE, MITRE). In MITRE ATT&CK, each tactic consists of different techniques an attacker may deploy. Techniques are more fine-grained steps attackers take to complete a tactic. In total,

MITRE ATT&CK consist of 10 tactics and 144 techniques (MITRE, MITRE) as provided on the MITRE attack website. First, we discuss the different attack kill chain phase and tactics present in the ATT&CK framework, and then discuss associated techniques.

MITRE Kill Chain Phase

Recent studies have shown that most attackers use repeated attack pattern behaviors in executing their attacks (Hutchins et al., 2011). By carrying out attacks in phases, attackers can methodically complete their attack mission without being detected by defenders. Knowledge of these attack tactics and techniques can help defenders emulate known attack strategies and evaluate their defense strength and weakness in detecting and stopping these attacks. These phases include ***reconnaissance, weaponization, delivery, exploitation, command and control, execution and maintenance***. The early phase of the attack lifecycle consists of the reconnaissance, weaponization, delivery, and exploitation, while the late phase consists of command and control, execution, and maintenance.

Tactics

These describe **why** an attacker exhibits a particular behavior on the victim's network or system infrastructure. MITRE ATT&CK tactics include ***persistence, privilege escalation, defense evasion, credential access, discovery, lateral movement, execution, collection, ex filtration, command and control***. Figure 5.5 shows an example snapshot of MITRE ATT&CK tactics and their corresponding techniques (MITRE, MITRE). For example, the attacker may deploy the **Discovery** tactic to discover network or system resources of interest on a victim's infrastructure, and deploy techniques such as **account discovery**, **network service scanning**, **process discovery**, or **query registry** to complete the attack action.

Discovery	Lateral Movement	Collection
Account Discovery	Windows Remote Management	Clipboard Data
File and Directory Discovery	Remote File Copy	Data from Network Shared Drive
Local Network Protocol Configuration Discovery	Remote Desktop Protocol	Email Collection
Network Service Scanning	Remote Services	Data from Protocol Removable Media
Process Discovery	Replication Through Removable DLL Injection Media	Input Capture
Query Registry	Windows Admin Shares	Screen Capture

Figure 5.5: MITRE ATT&CK Matrix Snapshot

Figure 5.4 shows the description of the discovery/internal reconnaissance tactic from the APT1 report (Fireeye, a). The APT1 report details the attack steps used by Chinese cyber attackers on US government agencies and military contractor companies. This tactic description shows how the attackers leveraged network discovery tools to list victim's network configurations, running processes, user accounts, administrator accounts, and network connections.

Techniques

Techniques describe **how** an attacker intends to complete an attack tactic. For example, to complete a persistence attack, an attacker may use "installing bootkit technique". By installing a bootkit, the attacker installs malware in the Master Boot Record (MBR) of the operating system such that the malware always executes before the operating system or antivirus software, resulting in the attacker evading the detection mechanism while persisting on the victim's system. In addition, the MITRE ATT&CK framework provides mitigation for

Table 5.1: Russian Hammertoss Attack report generated by FireEye Security Company with corresponding Tactics and Techniques categories.

Attack Steps	Description	Tactics and Techniques
1.Hammertoss looks for twitter handle	The HAMMERTOSS backdoor generates and looks for a different Twitter handle each day. It uses an algorithm to generate the daily handle, such as "234Bob234", before attempting to visit the corresponding Twitter page	**Tactic**: Defense evasion **Technique**: Redundant access, valid accounts
2.Hammertoss retrieves a url with location of image data	. HAMMERTOSS visits the associated Twitter account and looks for a tweet with a URL and a hashtag that indicates the location and minimum size of an image file.	**Tactic**: Defense evasion **Technique**: Redundant access, valid accounts
3.Hammertoss retrieves the image with malicious content	HAMMERTOSS visits the URL and obtains an image.	**Tactic**: Command & Control **Technique**: Web Service, Data Obfuscation
4.Hammertoss retrieve data from image and decrypts to obtain commands	The image looks normal, but actually contains hidden and encrypted data using steganography. HAMMERTOSS decrypts the hidden data to obtain commands	**Tactic**: Command & Control, Defense evasion **Technique**: Data encoding, Obfuscated Files or Information

each technique which recommends how to defend against the attack technique. For example, to defeat the "installing bootkit technique", the framework offers that trusted computing can be leveraged to ensure secure and trusted boot process. Figure 5.5 shows the techniques that may be employed depending on the tactics and intent of the attacker.

5.4.2 Sample Threat Report with Tactic and Technique Categorization

An example threat report generated by a security organization with ATT&CK categories is illustrated in Table 5.1 for clarity. It describes the Hammertoss advanced persistent attack used by Russian hackers. Hammertoss leveraged different attack tactics and techniques to deliver attack payload by using a combination of known attack steps in an ingenious way, making their attacks very potent.

In step 1 and 2 of the attack, Hammertoss attackers used Twitter handles to pass information to already compromised hosts on the victim's network. By leveraging *defense*

evasion tactics to evade detection and using the *Redundant access, valid accounts* technique, the attackers used a legitimate service (Twitter), which are allowed on most enterprise networks, to deliver attack packets. In step 3 the attacker retrieved the payload image using the *Command and Control* tactic leveraging the *Web Service and Data Obfuscation* technique to conceal command data in images. The command retrieved from the obfuscated image is then used to steal user information over the network. In step 4, the attack command was extracted from the image and decrypted using *Command and Control, Defense evasion* tactic using *Data encoding, Obfuscated Files or Information* technique before ex-filtrating the stolen data.

5.4.3 Bias Correction

Sampling bias is a known problem in natural language processing (Jiang and Zhai, 2007) (Dong et al., 2018), where the training data distribution is often biased compared to the test data distribution. With regard to threat report classification, a bias in training data may occur if we can only obtain reports with appropriate labels from a small number of security organizations whose formatting styles are not a good representation of the population. Here, the labeled data can be used to train a machine learning classifier. Given a set of reports from various organizations for category prediction, the challenge is to address bias in labeled data distribution so that the classifier generalizes well on reports from all organizations. In this work, we utilize a bias correction technique to address a type of bias called covariate shift.

Covariate Shift

This is a form of transfer learning that relates the training and test distribution which are unequal, but obtained from the same domain. Particularly, the relation between training and test distribution occurs with an equality in class conditional distribution while the

covariate distribution is unequal. This is called sampling bias (Huang et al., 2007). Concretely, let $p_{tr}(\mathbf{x}, y)$ and $p_{te}(\mathbf{x}, y)$ be the joint probability distribution of training and test data respectively where $\mathbf{x}$ is a d-dimensional data instance whose class label is denoted by y. The covariate shift assumption is that $p_{te}(y|\mathbf{x}) = p_{tr}(y|\mathbf{x})$ while $p_{te}(\mathbf{x}) \neq p_{tr}(\mathbf{x})$. Using this assumption, one can correct bias in training data by evaluating an instance weight $\beta(\mathbf{x})$ such that $\beta(\mathbf{x}) = \frac{p_{te}(\mathbf{x})}{p_{tr}(\mathbf{x})}$ for each training instance $\mathbf{x}$. Unfortunately, estimating $p_{te}(\mathbf{x})$ and $p_{tr}(\mathbf{x})$ separately is NP-Hard. So, recent studies have focused on developing techniques for computing the density ratio directly. These include Kernel Mean Matching (KMM) (Huang et al., 2007), Kullback-Leibler Importance Estimation Procedure (KLIEP) (Sugiyama et al., 2008), and unconstrained Least Square Importance Fitting (uLSIF) (Kanamori et al., 2009). In this study, we utilize all three of these bias correction methods.

Kernel Mean Matching

In kernel mean matching, the mean Euclidean distance in a Reproducing Kernel Hilbert Space (RKHS) between the data distribution of the weighted training data and the corresponding distribution of the test data is minimized. The mean distance is calculated by evaluating the *Maximum Mean Discrepancy* (MMD), which is represented as $\left\| E_{\mathbf{x}\sim p_{tr}(\mathbf{x})}[\beta(\mathbf{x})\phi(\mathbf{x})] - E_{\mathbf{x}\sim p_{te}(\mathbf{x})}[\phi(\mathbf{x})] \right\|$, where $\|\cdot\|$ represents the l_2 norm, and $\mathbf{x}$ is a data instance in any given dataset $\mathbf{X}$. (Yu and Szepesvári, 2012).

The empirical approximation of the MMD to obtain the desired $\hat{\beta}(\mathbf{x})$ is given by the quadratic program,

$$\hat{\boldsymbol{\beta}} \approx \underset{\beta}{\text{minimize}} \frac{1}{2}\boldsymbol{\beta}^T \mathbf{K}\boldsymbol{\beta} - \boldsymbol{\kappa}^T\boldsymbol{\beta} \tag{5.1}$$

KLIEP

By leveraging a density ratio estimation scheme called Kullback-Liebler Importance Estimation Procedure (KLIEP) (Sugiyama et al., 2008), KLIEP performs online updates to

importance weights by minimizing the KL-divergence between the weighted source and target distribution. For each new instance, the importance weights are updated online.

Relative Density Ratio Estimation

By leveraging relative densities between the training and test data distributions, Yamada et at. (Yamada et al., 2011) presented an importance weighting scheme for covariate shift correction. Instead of computing the density ratio $\beta(x)$ in KMM and KLIEP, they leverage the use of relative density ratio. If $P(x)$ and $P'(x)$ are the densities of test and training distributions respectively, then $\beta(x) = \frac{P(x)}{P'(x)}$. In some cases where $P'(x)$ is small, the density ratio β results in large values, resulting in unreliable divergence estimate. To overcome this limitation, the relative density ratio is defined as follows. For $0 < \alpha < 1$, the α-mixture density of $P(x)$ and $P'(x)$ is defined by,

$$P_\alpha(x) = \alpha P(x) + (1 - \alpha)P'(x) \tag{5.2}$$

Therefore, the α-relative density ratio is given by $r_\alpha(x) = \frac{P(x)}{P_\alpha(x)}$. Similar to KLIEP, the α-relative density ratio can be directly computed using a parametric model given by

$$\hat{r}_\alpha(x) = \sum_{i=1}^{N} \hat{\theta}_i K_\sigma\left(x, x^{(i)}\right) \tag{5.3}$$

where $\hat{\theta}_i$ is the i^{th} parameter to be learned from data, and K_σ is the Gaussian kernel function with width σ. By leveraging this parametric form of α-relative density ratio, the PE divergence between weighted training and unweighted test data distribution can be minimized.

5.5 Approach

Figure 5.6 shows the overview of our approach. First, we gather threat reports from different security organizations around the world. Our data sources include threat reports scraped from websites in both HTML and Adobe PDF formats. Second, the data is parsed and

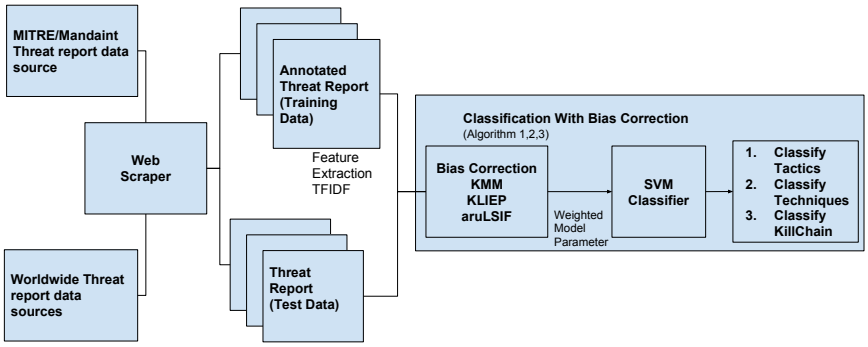


Figure 5.6: Threat Report Classification System

Algorithm 1: Algorithm Illustrating Our Approach

Input: Document Urls: urls
Result: Tactics and Techniques Classifier Accuracy

```
1 begin
2     docs = Download_Documents(list of urls)
3     features = Extract_features(docs) /*using alg 2*/
4     Apply_BiasCorrection(features) /*using alg 2 for bias correction*/
5     classifyTactics() /*using algorithm 2*/
6     classifyTechniques() /*using algorithm 3*/
7     return TacticsandTechniquesAccuracy
```

extracted to plain text format. Third, using NLP techniques, we extract TF-IDF features from each document. After feature extraction in both training and test data, we train a bias-corrected classifier by first estimating importance weight of training data using KMM, KLIEP, aruLSIF and then training a classifier with weighted training data.

The listing for algorithm 1 shows the full implementation steps. Algorithm 2 shows the pseudo code for the tactics classification task. For this alogrithm, we take the data input as a set of training data and test data. In line 2 and 3, we randomly sample and split the test data to validation and test set. We then extract useful words using TF-IDF to extract features for each document as shown in line 4,5 and 6. After generating the features, we obtain training, validation and testing data matrix.

In line 7, we generate the estimating density ratio $\beta(\mathbf{x})$ by initializing it to zero. For estimating density ratio $\beta(\mathbf{x})$, we apply the radial basis function on training and test data to

Algorithm 2: Threat Report Classification with Bias Correction

Data: $\mathbf{Doc}_{tr}$, $\mathbf{Doc}_{te}$, Covariates $\mathbf{X}_{tr}$ and $\mathbf{X}_{te}$
Input: Sample Size: m, Tolerance: η, Parameters: θ
Result: *Accuracy*

1 **begin**
2 $\quad \mathbf{X}_{val} \leftarrow GetValidationData(\mathbf{Doc}_{te})$
3 $\quad \mathbf{X}_{te1} \leftarrow GetTestData(\mathbf{Doc}_{te})$
4 $\quad \mathbf{X}_{tr} \leftarrow extractTFIDF(\mathbf{Doc}_{tr})$
5 $\quad \mathbf{X}_{val} \leftarrow extractTFIDF(\mathbf{Doc}_{val})$
6 $\quad \mathbf{X}_{te1} \leftarrow extractTFIDF(\mathbf{Doc}_{te1})$
7 $\quad \hat{\boldsymbol{\beta}} \leftarrow zeros$
8 $\quad \mathbf{X}_{tr}^{*(i)} \leftarrow generateSample(\mathbf{X}_{tr}, m)$
9 $\quad \hat{\boldsymbol{\beta}}^{*(i)} \leftarrow BiasCorrection(\mathbf{X}_{tr}^{*(i)}, \mathbf{X}_{val}, \theta)$ /*Equation 5.1*/
10 $\quad \hat{\boldsymbol{\beta}} \leftarrow aggregate(\hat{\boldsymbol{\beta}}^{*(i)})$
11 $\quad \mathbf{Result}_{Acc} \leftarrow SVM(\mathbf{X}_{tr}, \mathbf{X}_{te1}, \hat{\boldsymbol{\beta}})$
12 $\quad$ **return** *Accuracy*

obtain the kernel matrices. To learn the importance weight, the training and the validation data is passed to theBiasCorrection method in line 9. In line 11, the learned weighting parameter is passed to the SVM classifier to correct distribution bias in training data to learn a bias-corrected model and evaluate classifier accuracy using the remaining part of the test data.

Classification of Techniques with Confidence Propagation

The goal of this chapter is to predict both tactics and techniques for a given attack description. For example, a typical threat report can be labeled as **defense evasion** at a macro-level, i.e., at the tactics-level, and as **bootkit installation** at a more fine-grained micro-level label for techniques. After predicting the tactics, we need to predict the techniques used by the attacker pertaining to the predicted tactic. Therefore, we apply a confidence propagation algorithm (Wu et al., 2004) that utilizes the confidence of the tactic classifier as a prior towards computing the technique class.

Algorithm 3: Confidence Score Propagation for Technique Classification

Input: Input level2 data: s, Level 2 class: $\mathbf{C}_i$
Tactic Confidence scores: $\{\mathbf{C}_j ... \mathbf{C}_k\}$
$p_{te}(s|\mathbf{C}_i)$/* The confidence score of s belonging to technique Ci*/
$p_{ta}(s|\mathbf{C}_j)$/* The confidence score of s belonging to tactic Cj*/
Result: $p_{te}(s|\mathbf{C}_i)$/* Updated confidence score of s belonging to technique Ci*/

1 **begin**
2 **for** $\mathbf{C}_m \in \{\mathbf{C}_j ... \mathbf{C}_k\}$ **do**
3 /*Check if Cm is an parent of Ci, if it is modify the confidence score of Ci*/
4 $\lambda_m^i = 1/(exp(abs(p_{te}(s|\mathbf{C}_i) - p_{ta}(s|\mathbf{C}_m))))$
5 $p_{te}(s|\mathbf{C}_i) = p_{te}(s|\mathbf{C}_i) + \lambda_j^i * p_{ta}(s|\mathbf{C}_j) + .. + \lambda_k^i * p_{ta}(s|\mathbf{C}_k)$

Due to the nature of available labeled data, we observe that the number of training data instances per class for macro-level tactic classification is significantly higher than the number of training instances per class available for micro-level technique classification. This long tail class distribution for micro level classes typically result in poor classifier performance, in terms of accuracy and confidence, compared to the macro-level label prediction. By propagating the classifier confidence score from the macro-level classifier to the classifier confidence of micro-level classes, we can improve the overall classification accuracy. We achieve this by first computing a scaling factor using classifier confidence between each class label pair of micro-level and macro-level classes, and then applying this to obtain a boosted classifier confidence on the micro-level class predictions.

We first train a micro-level classifier to classify a given text into different techniques. Then, we utilize the confidence propagation method to boost confidence score for techniques associated with the predicted tactic. Algorithm 3 details the confidence propagation method. We provide the classifier confidence score for the tactics classes as input. Then, for each technique, we obtain the output of micro-level classifier's initial confidence score. This is used to compute the scaling factor as shown in line 2 where for each of the tactics classifier confidence score $\{\mathbf{C}_j ... \mathbf{C}_k\}$, the scaling factor λ_m^i is calculated to determine the influence of the tactics class on the technique class. The scaling factor λ_m^i increases with tactics with

larger influence as shown in line 4 where λ_m^i is inversely proportional to the exponential of the difference of p_{ta} from p_{te}. We then use the corresponding scaling factor to modify the confidence of the classifier for the fine-grained micro-level classes as shown in line 5.

The confidence propagation approach improves the accuracy at the technique-level because it allows us to leverage the known tactics class to boost the classifier confidence score at the technique-level. For example, if there are two techniques that have similar confidence scores at the technique-level, but one of the techniques belong to a tactics class with a higher confidence score, then by propagating the confidence score of that tactics class to the corresponding child technique class, the confidence score of the corresponding child technique increases, therefore improving the accuracy of the classifier.

5.5.1 Kill-Chain Phase Detection

After classifying to tactics, we map the corresponding tactics to the kill chain phases. In order to generate the kill-chain phase mapping to tactics, we leverage automated rule extraction techniques which include the decision tree method (Quinlan, 1986) and the ripper algorithm (Cohen, 1995b). To generate these rules, we labeled the documents with the corresponding tactics and killchain phases present in the documents. Using this as training data, we ran the decision tree and ripper algorithm to learn the rules and generate the rules. We evaluated the rules by comparing the automatically generated rules with the manually mapped rules.

Table 5.2 shows some of the rules extracted using decision tree and ripper algorithm. For example, rule 1 extracted by the decision tree can be interpreted as follows, if Exfiltration or command and control tactics is not present, then the kill-chain phase is maintain. Rule 2 specifies if command and control tactic is present, then the kill-chain phase is control. The ripper algorithm extracts similar rules as the decision tree rule extraction method. For ripper rule extraction, rule 1 states that if command and control tactic is present then the kill-chain

Table 5.2: Automated and Manual Rules Generated From Data

Rule Generator	Rules
Decision Tree	Rule 1: If $\neg Present(Exfiltration) \wedge$ $\neg Present(CommandandControl)$ $\Rightarrow Killchain(maintain)$ Rule 2: If $Present(CommandandControl)$ $\Rightarrow killchain(control)$ Rule 3: If $Present(exfiltration)$ $\Rightarrow killchain(execute)$
Ripper	Rule 1: If $Present(CommandandControl)$ $\Rightarrow killchain(control)$ Rule 2: If $\neg Present(CommandandControl)$ $\Rightarrow killchain(maintain)$
Manual	Rule 1: If $Present(CommandandControl)$ $\Rightarrow killchain(control)$ Rule 2: If $\neg Present(CommandandControl)$ $\Rightarrow killchain(maintain) \vee killchain(execute)$

phase is control which is similar to rule 2 for decision tree rule extraction method. Rule 2 states that if command and control tactic is not present, then the corresponding kill-chain phase is maintain. To compare if the automated rules are correct, we manually extracted some of the rules. We can observe that rule 1 extracted using manual extraction is similar to rule 2 and rule 1 extracted by decision tree and ripper algorithm respectively.

After classifying to kill-chain phase, tactics and techniques, we can extract the corresponding mitigation that can be leveraged to defeat the attacker's strategy.

5.6 Evaluation

5.6.1 DataSet

Table 5.3 shows the summary of the statistics of the two datasets we considered to evaluate our framework. We downloaded 169 labeled documents from Mitre ATT&CK website which is denoted as **Att&ck dataset**. In addition, we downloaded 17,600 documents from the Symantec threat report website, which is denoted by **Symantec dataset**. To fur-

Table 5.3: Statistics of Dataset

Dataset	Average File Size(KB)	Number of Documens	Average Words/Document	Total Size(MB)
Att&ck	2	169	1000	0.3
APTReport	50	488	10,000	1000
Symantec	5	17,600	900	147

Table 5.4: Tactic Classification Accuracy Result for **APTReport dataset** and **Symantec dataset**

Training	Test	Validation	TTPDrill	SVM	KMM	KLIEP	ARULSIF
APT	SYM	SYM	14.60	59.6	58.20	**60.20**	59.40
APT	SYM	-	15.90	61.1	61.20	**61.20**	60.20
APT +ATT	SYM	SYM	13.80	62	61.00	60.20	**64.00**
APT + ATT	SYM	-	16.80	61	48.30	60.80	**61.20**
SYM	APT	APT	39.50	**93.6**	67.00	90.10	90.10
SYM	APT	-	50.00	91.1	90.93	**96.30**	96.00
SYM + ATT	APT	APT	14.80	90	92.20	92.50	**93.00**
SYM + ATT	APT	-	63.90	91	**96.30**	96.30	96.30

∗APT: **APTReport dataset** only
∗SYM: **Symantec dataset** only
∗APT+ATT:Union of **APTReport dataset** and **att&ck data**
∗SYM+ATT: Union of **Symantec dataset** and **att&ck data**

ther evaluate our approach on threat reports with more complicated sentence structures, we downloaded 488 documents containing reports of highly sophisticated advanced persistent threat attacks from security companies including FireEye, Mandiant, Mcaffe and other various Anti-Virus Companies. We denote this by **APTReport dataset**. The total size of the dataset considered is 300KB for **Att&ck dataset**, 147MB for **Symantec dataset** and 1GB for **APTReport dataset**. We obtained threat reports on data breaches, including the stuxnet breach, Target credit card data breach report, Home Depot data breach, Chinese APT campaign on US government and companies, Russian APT attacks, lazarus ransomeware reports, and shamoon attack on oil refineries to mention a few.

Feature Extraction : To evaluate our approach, For the Symantec dataset, we manually labeled all the 17,600 documents and used the 169 documents from the **Att&ck dataset** to validate our model. we extracted 200 TFIDF features from 17,769 documents, resulting in a total of 17,769 instances. Similarly, for the **APTReport dataset**, we manually labeled all the 488 documents and added the 169 documents from **Att&ck dataset**. This resulted in a total of 657 instances with 200 features.

Classification : For our evaluation, we used data from different sources as training and testing data. We considered various experimental setups to evaluate our approach. First, we used **Symantec dataset** as training data and **APTReport dataset** as test data. Second, we used **APTReport dataset** as training data and **Symantec dataset** as test data. Third, we combined the **Att&ck dataset** with the training data portion. Fourth, one may argue that using the whole test data for learning the weight may not be realistic since all the test data is not available at training time. To overcome this challenge, instead of using all the test data as the target data to learn the importance weight for bias correction, we split the test data to validation and test data. The validation data is used the learn the weight for bias correction and the second portion of the test data is used to evaluate the classifier accuracy.

Each of the pairs of training and test data is then used for estimating importance weight for bias correction, and then is provided to an SVM classifier for training. Our implementation consists of 600 lines of python code. We leverage python data processing pipeline and sci-kit learn module to perform data preprocessing and feature extraction. Finally, we implement the QP of KMM using cvxopt python module.

5.6.2 Tactics Classification Results

Table 5.4 shows the result for tactics classification result for all combinations of the different experimental setup. we can observe that in all cases TTPDrill performs very poorly compared

to when we applied our bias correction method. We obtained better accuracy with all the bias correction method compared to the TTPDrill method. Using **APTReport dataset** as training data, the classifier accuracy for TTPDrill does not perform better than 17% while the classifier accuracy exceeds 58% for all bias correction methods. Likewise, by using **Symantec dataset** as training data and **APTReport dataset** as test data, we obtained a maximum of 63% and a worst performance of 14.8% for TTPDrill while we obtained more than 90% classifier accuracy with all bias correction methods except for 67% for the KMM method.

From our results, we can observe that using the whole test data to perform a bias correction performs better than using a portion of it as validation set to learn the weight. This is because, using all the test data provide better representation of the distribution of the target data distribution for bias correction. In addition, our approach performs better when we used **Symantec dataset** as training data and **APTReport dataset** as test data because **Symantec dataset** has more instances therefore allowing the classifier to learn a better classification model. Our approach works, since we take into consideration the bias in the data from different sources which TTPDrill does not consider.

5.6.3 Kill Chain Phases Classification Results

After classifying tactics, we used the ripper rules discussed in subsection 5.5.1 to map tactics to the corresponding kill-chain phase. As shown in Table 5.5, similar to the tactics classification results, our bias correction method outperforms TTPDrill for all experimental setup. Using **APTReport dataset** as training data and **Symantec dataset** as test data, we obtained 60.20 % classifier accuracy with the KLIEP bias correction method when we obtained 40.40% for TTPDrill method. Likewise, Using **Symantec dataset** as training data and **APTReport dataset** as test data, we obtained 96.3% classifier accuracy with the KLIEP bias correction method while we obtained 52.70% for TTPDrill method. From our results, our bias correction method outperforms TTPDrill significantly.

Table 5.5: Kill Chain Classification Accuracy Result for **APTReport dataset** and **Symantec dataset**

Training	Test	Validation	TTPDrill	SVM	KMM	KLIEP	ARULSIF
APT	SYM	SYM	40.40	59.6	58.20	**60.20**	59.40
APT	SYM	-	39.60	61	**61.20**	61.20	60.20
APT + ATT	SYM	SYM	35.40	62	**64.60**	60.20	64.00
APT + ATT	SYM	-	40.50	61	54.20	60.80	**61.40**
SYM	APT	APT	45.20	90	68.70	**90.10**	90.10
SYM	APT	-	52.70	93.8	91.20	**96.30**	96.00
SYM + ATT	APT	APT	45.56	91	92.20	92.50	**93.00**
SYM + ATT	APT	-	66.60	91.1	**96.30**	96.30	96.30

∗APT: **APTReport dataset** only
∗SYM: **Symantec dataset** only
∗APT+ATT:Union of **APTReport dataset** and **att&ck data**
∗SYM+ATT: Union of **Symantec dataset** and **att&ck data**

5.6.4 Techniques Classification Results

Figure 5.7 compares classifier accuracy for technique prediction with and without confidence score propagation. We use the K-nearest neighbor classifier due to the limitation of having few instances per technique class since the technique class consist of 144 classes. By applying confidence propagation using 20% of the tactics class, we obtained 8% classifier accuracy while we obtained 2% classifier accuracy without confidence score propagation. We obtained 23.9 % accuracy by propagating 40% of the tactics class and 18% without confidence score propagation. By propagating only 60% of the tactic classes, we obtain 45% classifier accuracy while we obtain 30% classifier accuracy without confidence score propagation. In addition, propagating 80% of tactics classes, we obtain 72% classifier accuracy while we obtain 49% classifier accuracy without confidence score propagation. Lastly, by propagating all the tactics level classes, we obtain 86% classifier accuracy with confidence propagation and 71% classifier accuracy without confidence score propagation. By applying confidence score propagation, the average classifier accuracy for technique prediction increased by 13%.

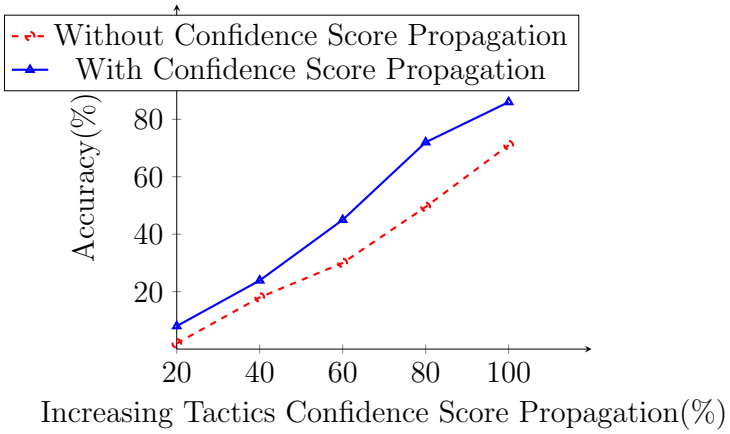


Figure 5.7: Technique Classification Accuracy With Confidence Score Propagation using **APTReport dataset**

CHAPTER 6

DECENTRALIZED IOT DATA MANAGEMENT [1] [2]

6.1 Introduction

With the advancement in embedded processors, actuators, sensors and communication systems, everyday devices are retrofitted with capabilities to communicate, compute and complete automated tasks (Fernandes et al., 2016; Jia et al., 2017). For instance, many of our everyday appliances have been retrofitted with capabilities to connect to the Internet (Gubbi et al., 2013). Such IoT devices include smart pacemakers, heart rate monitors, smart refrigerators, smart coffee makers, smart television, smart home assistants, and smart door locks. By equipping these devices with computational and communication capabilities, these devices collect and transmit large amounts of privacy-related data (Bertino, 2016). For example, IoT devices such as smart cameras, smart health monitoring devices (Williams and McCauley, 2016) such as heart rate monitors, glucose level monitors can reveal private information about the users.

Due to the limited processing capabilities of IoT devices (Gonzalez et al., 2016; Masud et al., 2008), IoT devices usually leverage externally controlled third-party service providers to perform additional data processing. By transmitting sensitive user data to third-party services providers (Fernandes et al., 2016), users are forced to trust service providers to enforce data protection and provide data privacy guarantee. Unfortunately, service providers

[1] This chapter contains material previously published as: Gbadebo Ayoade, Vishal Karande, Kevin Hamlen, and Latifur Khan. "Decentralized IoT Data Management Using BlockChain and Trusted Execution Environment." In *Proceedings of the 19th IEEE International Conference on Information Reuse and Integration for Data Science (IRI)*, pp. 15–22, July 2018. Ayoade led the design, implementation and evaluation of the approaches used in this research.

[2] This chapter contains material previously published as: Gbadebo Ayoade, Amir El-Ghamry, Vishal Karande, Latifur Khan, Mohammed Alrahmawy, Magdi Zakria Rashad. "Secure data processing for IoT middleware systems", *The Journal of Supercomputing, 2018.* Ayoade led the design, implementation and evaluation of the approaches used in this research.

often violate data privacy policies by using data collected from users for unauthorized purposes (Hu et al., 2011). This undue advantage by service providers is based on centralized architecture where trust in a third party system as a central authority is required to manage user data. To eliminate this imbalance in data access policy enforcement between service providers and users, we propose a system of decentralized data management using a decentralized asset management system based on Blockchain (Nakamoto, 2009) and smart contract technology (Foundation, 2014).

With the advent of decentralized asset management systems as seen in the finance sector which leverages blockchain technology such as seen in Bitcoin (Nakamoto, 2009), electronic fund transfer can occur without the need for a centralized electronic fund management system. With this technology, money transfer can occur across international boundaries without the bureaucracy of centralized authorities. Due to the decentralized nature of blockchain technology, proposed applications (Foundation, 2014) in various fields include automated insurance management, supply chain management, decentralized commercial data storage as seen in Filecoin (Filecoin, 2017). For instance, Slock It (slock, 2017) uses blockchain to provide automated device sharing platform for IoT devices such as smart locks.

By leveraging this decentralized architecture, we propose a system that limits the authority of centralized data management systems. Blockchain technology (Nakamoto, 2009) and smart contracts (Foundation, 2014) allow decentralized management of data among *untrusted* parties called miners. Blockchain (Nakamoto, 2009) is a distributed ledger where transaction state integrity is enforced by distributed consensus among decentralized untrusted parties. To enforce the integrity of the blockchain, each current block generated by the miners must contain a hash of the previous block in the blockchain as shown in Figure 6.2, making it difficult to modify the transactions recorded in the blockchain.

Smart contracts (Zyskind et al., 2015) are autonomous applications that run within the blockchain. With smart contracts, we provide a system where *rules* that govern interactions

among interested parties is enforced autonomously in the blockchain network without a centralized trust. By leveraging this capability, we can equip the users with the capacity to control how their data is accessed and used since a smart contract provides them with equal data management privilege. Furthermore, smart contracts execute in isolated virtual machines on the miners' infrastructure. By using isolated virtual machines to run these smart contracts, miners cannot modify application outcomes. With smart contracts and blockchain, we can provide a data access audit system to track data usage among the interested parties leading to proper data access accountability.

All data stored in the blockchain has to be public for the miners to be able to verify transactions (Kosba et al., 2016). Our proposed system overcomes these challenges by storing the hash of the encrypted data in the blockchain, while the main data is encrypted and stored using trusted computing. By leveraging trusted computing, we can verify the integrity of the system used in our data storage. In our case, we use trusted computing as implemented by Intel SGX architecture.

By leveraging a trusted execution environment based on Intel SGX, we provide data protection from unauthorized access from powerful adversaries. SGX offers hardware-level protection of user data by enforcing process isolation by executing the programs in secure enclaves and protecting the enclave's memory pages by the CPU hardware. These secure containers called enclaves are protected from the operating system, other processes and hypervisor processes (Costan and Devadas, 2016).

In this chapter (Ayoade et al., 2018) (Ayoade et al., 2019), we make the following contribution.

- We leverage the blockchain platform to provide decentralized IoT data access management.
- We leverage smart contracts to provide equal data access management privilege among IoT users and IoT service providers.

- We provide data storage using a trusted execution environment(Intel SGX) for secure data storage.
- We provide a full system implementation on a real blockchain platform using Ethereum smart contracts.

The rest of this chapter is organized as follows. §6.2 provides background on Blockchain, SGX and IoT system. §6.3 discusses the scope, case for blockchain and SGX and challenges and solutions encountered in deploying SGX based system. §6.4 provides the architecture of our system. In §6.5, we describe our implementation approach and §6.6 provides the evaluation of our approach. §6.7 and §6.8 provide discussion and related work respectively. Finally, §6.9 concludes.

6.2 Background

6.2.1 Overview of Architecture

In this section, we discuss a brief overview of our system components as shown in Figure 6.1. To provide decentralized management of data generated by IoT devices, we store the hash of the encrypted data generated in the blockchain and then store the data itself in an SGX enabled storage system. As a result, the blockchain manages the data access policy through the smart contract.

To access data, third party users will request permission to access data from the blockchain by utilizing the smart contract API. If request is granted, the hash of the data is returned and used to retrieve data from the SGX platform. Before the SGX platform retrieves that data from secure storage, it will independently recheck the blockchain for access permission before returning the data needed to the third party user. The intuition for these two step check is to ensure all access permission policy and authority is managed by the smart contract executing in the blockchain. The access check does not incur much overhead since

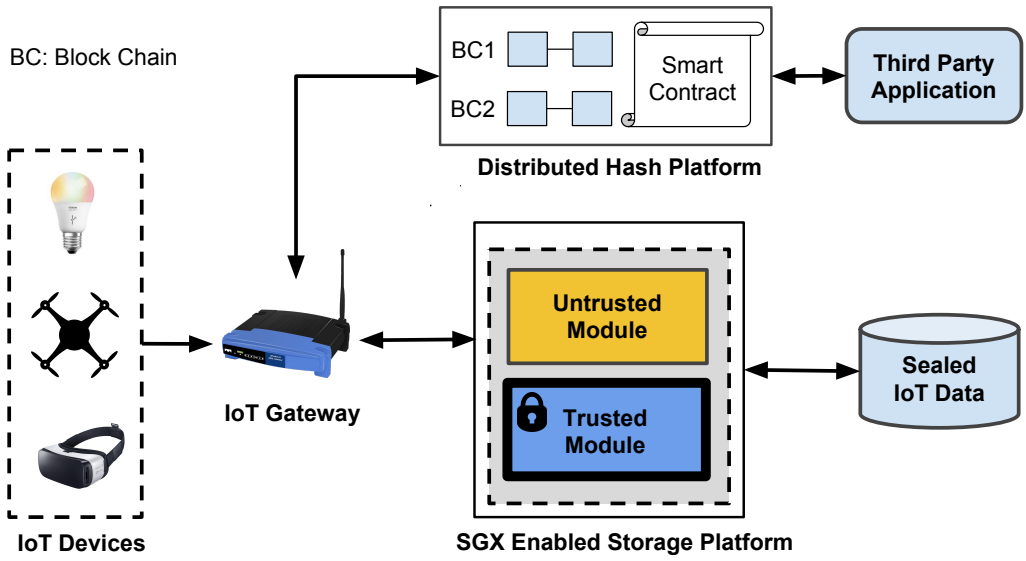


Figure 6.1: A Simplified Architecture of IoTSmartContract

access check is a read operation which has a fast execution time on the blockchain as we will later show in our evaluation section. In this section, we provide more background on the components of our IoTSmartContract system.

6.2.2 Internet of Things

With the increase in computational and communication capabilities and technological advancement in device miniaturization, every day devices are granted capabilities to sense and react to the environment through the use of sensors and actuators. A typical IoT architecture comprises of devices, sensors, actuators, IoT Hubs, IoT Gateway and a cloud service provider.IoT devices are devices with capability to sense and collect data which can be transmitted on a connected network for storage or further processing. IoT devices includes light bulbs, heart rate monitors, smart cameras and many more. With the IoT hub, different devices with disparate communication protocol such zigbee or bluetooth can connect to the IoT network. IoT networks includes IoT gateway which helps to provide data aggregation

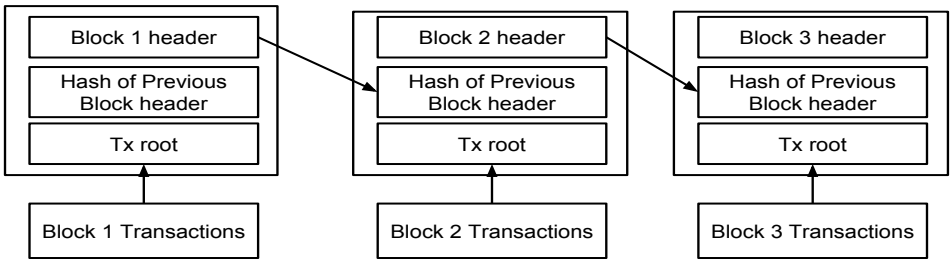


Figure 6.2: A BlockChain Data Structure

on the client network. To process the huge amount of data transmitted by the IoT devices, cloud services are used to store and further process the data.

6.2.3 BlockChain

Blockchain is a distributed ledger where the state of its transactions is maintained by a distributed consensus among untrusted entities without the need for a centralized trusted third party authority. These decentralized entities are called *miners* (Crosby et al., 2016; Nakamoto, 2009). By providing a proof of work, the miners bundle confirmed transaction in blocks by generating a hash of the current block which includes the hash previous block as seen Figure 6.2. These proof of work generation requires high computational CPU power, therefore protecting the blockchain from adversarial attacks.

The blockchain can store data and perform computations that can be executed by these decentralized entities to determine the state of the blockchain in an autonomous manner. These autonomous computations are called *smart contracts*. By leveraging smart contracts, we provide a system where *decentralized data access policy* control is enforced without relying on third party service providers, therefore ensuring continuous service delivery for IoT system users. We implemented the smart contract using Ethereum blockchain platform.

The *Ethereum* (Foundation, 2014) smart contract is an implementation of the smart contract with Turing complete computation. The Ethereum smart contract is deployed in

the blockchain and can be executed by the miners to determine the state of the program. By generating blocks, the miners can autonomously ensure that the state of integrity of the contract program.

In order to allow miners to run a deployed smart contract, the contract owner will pay the miners some fee called *Ethereum gas*. The higher the gas paid, the faster the speed of getting the contract to execute and generate confirmations on the blockchain. Because the smart contracts also store data, contract owners will need to provide gas for storage on the blockchain. In our case, we limited the data stored on the blockchain by storing only the the hash of the data and then encrypting the data and storing on another system.

To interact with a smart contract, each smart contract has a unique address in the blockchain. The address can be used to retrieve the contract and then get the ABI (Abstract Binary interface) which provides the API of the contract. By getting the smart contract API, a user can execute the smart contract API to perform some computation.

6.2.4 Trusted Execution Environment

Recent advancements in embedded hardware technology to support trusted execution environment (TEE) (e.g., TPM , ARM Trust Zone (Santos et al., 2014), AMD SVM (Van Doorn, 2006)), Intel SGX (Karande et al., 2017)) allow service providers to ensure confidentiality and integrity of data and computations by protecting code and data within a secure region of computation.

Intel SGX is a trusted computing architecture introduced in the new Intel Skylake processors. By providing a new set of instructions which extends the X86 and X86_64 architectures, user level applications can provide confidentiality and integrity without the trust of the underlying Operating System. With these instructions, application developers can create a secure and isolated containers called *enclave* to protect security sensitive computations. In particular, the memory content of an enclave is stored inside a hardware protected memory

region called as Enclave Page Cache (EPC). By leveraging the Memory Encryption Engine (MEE), all EPC pages are encrypted and any access to them is restricted by the hardware. Therefore, with SGX, applications can protect sensitive and secret data and computations from attacks from high privilege applications like the Operating system, hyper-visors and System Management Mode.

6.3 Overview

6.3.1 Scope and Assumptions

The scope of this chapter considers decentralization of data access management using blockchain and data privacy protection using Intel SGX. The main challenge is how to establish trust between IoT service providers and the users of IoT services. By leveraging smart contracts, we provide a data access management system where users have equal privilege in controlling how their data is shared or used. With smart contracts, we can specify data access rules that are autonomously enforced by untrusted third party entities on the blockchain network. For our platform, we assume all data is encrypted before transmission and all key exchange is performed using asymmetric cryptographic protocols (Rivest et al., 1978). In this chapter we do not consider replay attacks and denial of service attacks.

6.3.2 Threat Model

For our threat model, we consider the IoT data management service providers as untrusted entities since they have full control over user data, which give them undue advantage in how they use data or share user data with other third party entities.

Furthermore, we consider all third party users who request access to data to be untrusted. We assume all non data owner may leak data or use it for unauthorized purposes such as user's email for direct marketing.

We consider adversaries that seek to compromise the data storage cloud services by obtaining root privilege access to low level system resources such as memory, hard drives and Input/Output systems. These attackers employ techniques that compromise highly privilege applications such as Operating System and hyper-visors.

6.3.3 The Case of Using Blockchain for IoT data management

Decentralized Trust As users become more knowledgeable of data privacy leakage and its consequences, users may demand more control over how there data is being used. By leveraging blockchain, IoT vendors and service providers can provide services that users can trust since the data management system is done in publicly verifiable smart contracts program that run in the blockchain.

SmartContract Enforced Accountability With smart contracts, we can provide autonomous applications that enforce interaction rules among the system users without the need of centralized authority. Smart contracts allow individual entities with varied interest to generate rules that satisfy each participants interest. The rules are then programmed into smart contracts which is then enforced by the miners by independently verifying the state of the contract. For example, in centralized access policy management such Smartthings (Fernandes et al., 2016; Hue, 2017), if a user grants access or revoke access to their data, the user has to trust the third party service provider to comply and enforce his data restriction. With smart contracts, the users have equal privilege on how the policy is enforced since the policy enforcement is done by the miners on the blockchain network.

Audit Trail Enforcement By leveraging immutability of blockchain ledger (Nakamoto, 2009), we can provide immutable data access history of users' data. Since all entries in the blockchain is cryptographically linked to previous blocks generated on the blockchain, it is difficult for malicious attackers to modify the blockchain entries.

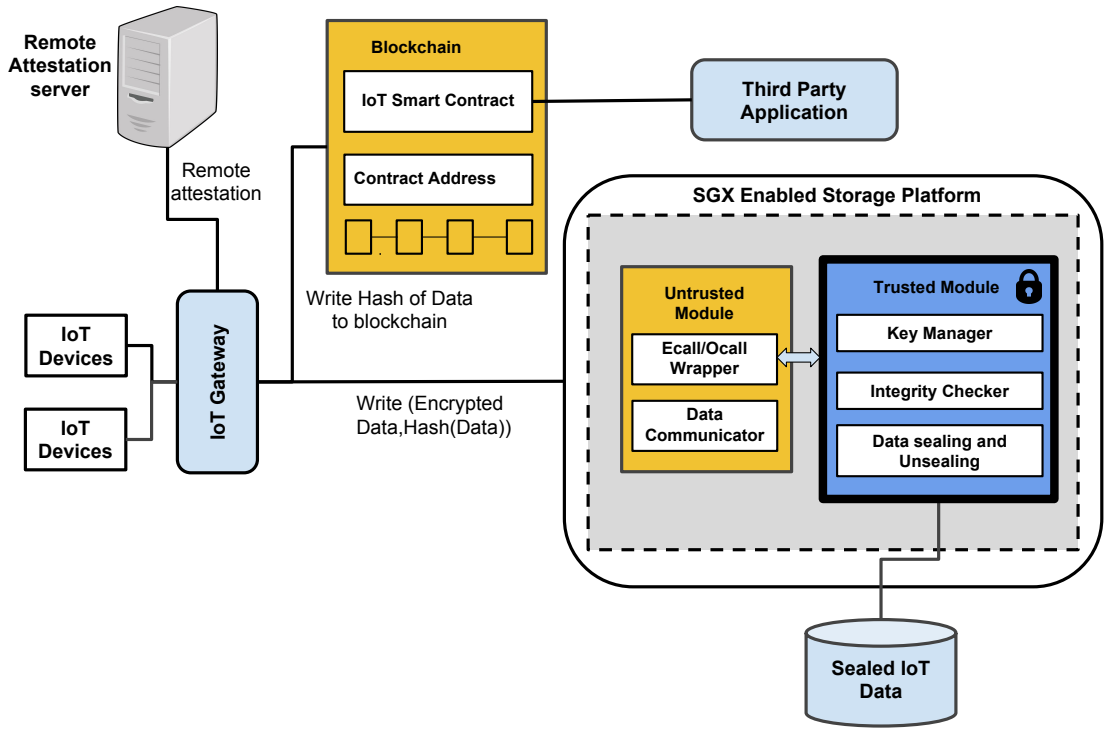


Figure 6.3: A IoTSmartContract Architecture

Algorithm 4: Smart Contract Pseudo-code

```
1: HashMap deviceRegistry(key:ownerAddress,value:List[DeviceIds])
2: HashMap deviceData(key:(ownerAddress,deviceId), value:List[DataHash])
3: HashMap DataAccessRegistry(key:(ownerAddresss,thirdpartyAddress,deviceId),value: bool isAllowed)
4: function RegisterDevice(ownerAddress,deviceID)
5:     InsertToHashMap(deviceRegistry)
6: end function
7: function WriteData(ownerAddress,deviceID,Data)
8:     if owner == ownerAddress
9:     deviceData[owner,deviceID].List.InsertData(hash(Data))
10: end function
11: function ReadData(ownerAddress,thirdPartyAddress,deviceID)
12:     if DataAccessRegistry(thirdPartyAddress) == true
13:     return deviceData[hash(ownerAddress,deviceID])
14: end function
15: function GrantAccess(ownerAddress,thirdPartyAddress,deviceID)
16:     if owner == ownerAddress
17:     DataAccessRegistry[hash(ownerAddress,thirdPartyAddress,deviceID)] = true
18: end function
19: function RevokeAccess(ownerAddress,thirdPartyAddress,deviceID)
20:     if owner == ownerAddress
21:     DataAccessRegistry[hash(ownerAddress,thirdPartyAddress,deviceID)] =false
22: end function
```

6.4 Architecture

As shown in Figure 6.3, IoTSmartContract consists of three main components which includes the IoT client network, the smart contract and the secure SGX module. The Client IoT network consist of all the IoT devices, the IoT gateway which connects the devices to the external network.

6.4.1 Smart Contract Component

As shown in algorithm 4, The Smart contract provides the decentralized access control policy to user data in form of Ethereum smart contract that executes in the blockchain. As a result of the limited data storage and fees required to store data in the smart contract, the smart contract only stores the hash of the data in the blockchain. The main data is encrypted and stored on the SGX module. The smart contract includes the user registration module, device registration, read and policy access module for hash data storage.

User Registration This module leverages the user registration system on Ethereum network. Each user joins the Ethereum network by generating a public private key pair which uniquely identifies the user. The private key can then be used to interact with the smart contract to perform functions such as device registration and data access.

Device Registration Each authenticated user can register their IoT devices by providing the identifier for the device. In the smart contract, we provide a hash map that maps the devices owned by a user to the owners address on the blockchain as denoted *mapping (address = list of owners deviceids)*

Data Write Access Policy For a device to write data to the blockchain, the device will provide the owners address and the device id with the data to be written. By using the

combination of the owner address and the device id as the key in a hash map, we can uniquely store all data that corresponds to all devices separately as denoted *((owner_address,device id) = list of device data)*. The value of the hash map is a list of hashes of the data written by the device. Before the smart contract allows data to be written to the contract, the smart contract will check if the owner address correspond to the device ID, so as to ensure only a device owner can execute write operation.

Device Data Read Access Policy For data access, a third party user who needs access to a device data from another user will request for permission to read the data. The requesting user will provide the address of the owner of the device and the device ID of the device. A hash map that contains the device owner and address and the device id as key with the list of the third party users as values is maintained within the smart contract. This is denoted as *((owner,device id,third party user address) = bool access)*. Before access is granted to the data, this hash map is checked to see if a requesting user can access the data by ensuring only registered third party users can access the device data.

6.4.2 IoTSmartContract Detailed DataFlow

In Figure 6.4, we show a detailed data flow diagram of IoTSmartContract. For a device to write or read data, first, the device communicates with the IoT gateway in **Step** ③ to register itself with the blockchain. For the IoT gateway to trust the SGX platform, it performs remote attestation as shown in **Step** ①. To perform data write, the device communicates with the IoT gateway in **Step** ②. The gateway then retrieves the smart contract address in **Step** ③. The gateway will then encrypt and hash the data. The hashed data will be written to the blockchain using the `writedata` function in the smart contract. The raw encrypted data is then written to the SGX platform in **Step** ④. By using the Ecall/Ocall wrapper, the untrusted module in the SGX application communicates with the trusted module as shown

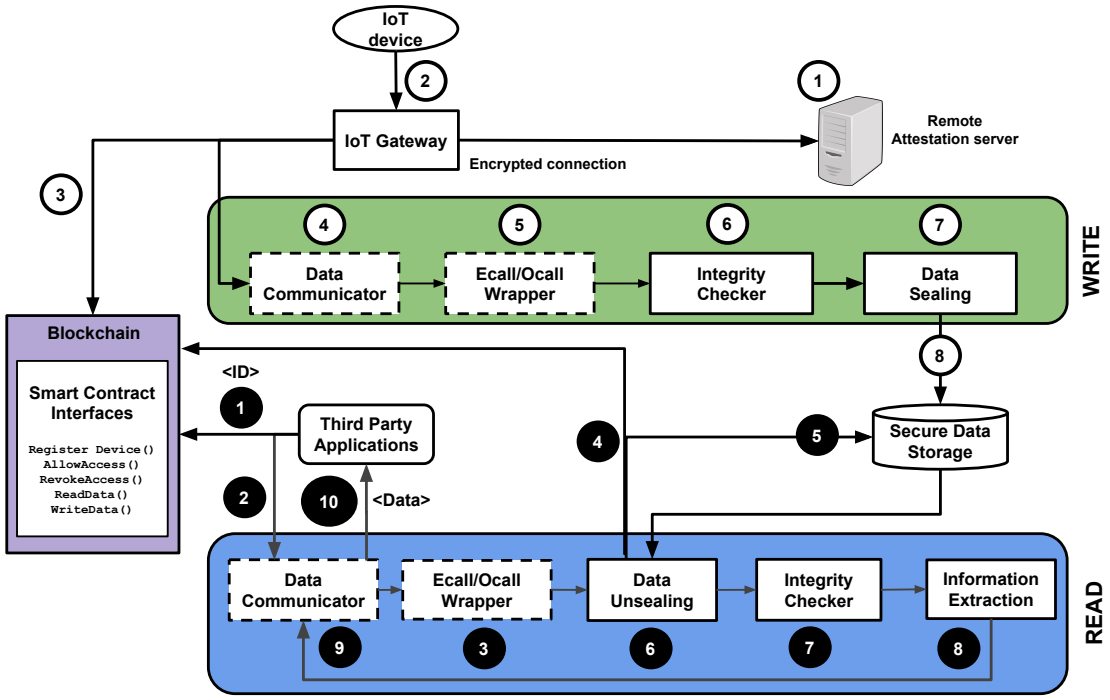


Figure 6.4: Illustration of the Data Flow in IoTSmartContract

in **Step** ⑤. In **Step** ⑥, the Integrity Checker module calculates the hash-based message authentication code(HMAC) of the data and appends the HMAC of the data before the data is sealed and written to disk in **Step** ⑦ and **Step** ⑧.

For the read operation, the user must register third party users with the smart contract by using the `allowAccess` method. To revoke access, the user calls the `revokeAccess` function. The third party user communicates with the smart contract as shown in **Step** ❶ to obtain the hash of the data generated by the device by supplying the device Id. The smart contract checks if the third party user can access the data from the device using the device Id and the address of the third party user, if permission is granted, the hash of the data is returned and can be used to access the data from the SGX storage platform. In **Step** ❹, the SGX application rechecks with the smart-contract using `READDATA` API to determine if the third party user can access the data hash identifier supplied by the third party request. If access is allowed, the SGX application retrieves the data from secure storage **Step** ❺ . Note

that the overhead for read operation from the blockchain is insignificant as we will show in the evaluation section in Table 6.1. The data is then unsealed in **Step ❻** and the Integrity Check **Step ❼** recalculates the HMAC of the data which is then compared with the stored HMAC. If the HMAC is unmodified the data is read and returned the the user as shown in **Step ❾** and **Step ❿**.

6.5 Implementation

We used five real IoT devices and a mobile phone to evaluate IoTSmartContract. The devices includes Philip Hue Hub with Zigbee light bulb, Samsung Smartthings Hub with Motion/Proximity sensor, Belkin Wemo Switch, Wemo Wall socket and a heart rate monitor mobile application on android.

6.5.1 Ethereum Smart Contract

We implemented the IoTSmartContract smart contract component using the Ethereum blockchain. Our implementation consists of 50 lines of code in solidity programming language. The code footprint needs to be concise so as to limit the amount of Ethereum gas needed to run a smart contract transaction. To limit storage space needed to store data in the blockchain, we only store the hash of the data. We ran the smart contract on the Rinkeby Ethereum test network for evaluation.

We implemented the following interfaces `registerDevice`, `allowAccess`, `writeData`, `readData` and `revokeAccess` that enable the IoT devices to interact with the smart contract . By using the geth Ethereum client, we can retrieve the smart contract address in the blockchain and performed operations such register devices, write data, read data, write and read access policy update and revoke access policy.

Table 6.1: Efficiency of Smart Contract Application based on Gas usage

Smart Contract Interface	Parameters	Gas Used
`registerDevice`	Device ID	47543
`allowAccess`	Device ID, ThirdParty Address	29517
`writeData`	Device ID, DataHash	51049
`readData`	Device ID, ThirdParty Address	–
`revokeAccess`	Device ID, ThirdParty Address	14792

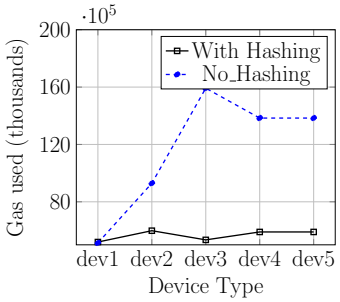


Figure 6.5: Gas utilization for Write Operation on Smart-Contract.

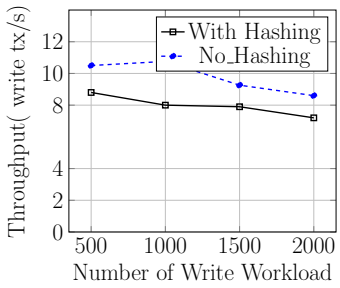


Figure 6.6: Throughput based on Increasing Write Workload

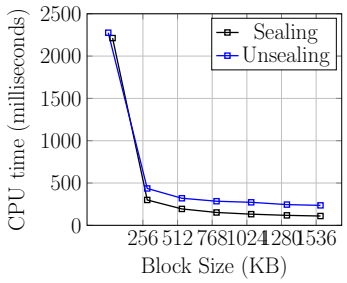


Figure 6.7: Avg Seal and Unseal time

6.6 Evaluation

Table 6.1 shows our evaluation result for each smart contract operation in gas used by the miners to complete an operation call. To confirm a transaction, the transaction must be included in a generated block. The data payload size for device 1 is 27 bytes, device 2, 47 bytes, device 3, 132 bytes, device 4 and 5, 127 bytes respectively while the hash length is 256 bits. As seen in Table 6.1, `registerDevice` uses 47,543 gas to complete its operation, `allowAccess` required 29,517 gas, `writeData` required 51,049 gas and `revokeAccess` required 14,792 gas. `readData` did not use any gas since reading from a smart contract is done on the local blockchain which does not require any mining.

In Figure 6.5, we compared the efficiency in gas usage required by miners to complete write operation in the blockchain. We compared two scenarios where the whole data which is encrypted from the devices is written to the blockchain versus writing only the hash of data. By considering 5 device types, we show that device 1 used 59,846 gas for hashed data compared to 159,234 gas required for raw data write which is a reduction of 169%. Device

2 used 53,454 gas for hashed data and 92,926 gas for raw write which gives a reduction of 73%. Device 3 and 4 used 58,974 gas for hashed data while 159,000 gas is required to write raw data which gives a reduction of 138%.

In Figure 6.6, we show the impact of increasing write workload on the blockchain. By increasing the write workload between 500 write requests to 2000 write requests, we measure the transaction throughput per second on the blockchain. Without hashing, For 500 write workload, the write transaction throughput is 10.56 writes per second. For 1500, it is 9.26 and for 2000 writes, the write throughput is 8.6 writes per second. With hashing, for 500 write workload, the write transaction throughput is 8.8 writes per second. For 1500, it is 7.9 and for 2000 writes, the write throughput is 7.2 writes per second. The write throughput decreases with increasing write workload. The write throughput also decreases with hashing enabled. Even though from Figure 6.5, the gas used with hashing enabled is constant because the hashing function produces 256 bit data for storage on the blockchain, the write throughput is lower because of the hashing process before writing the data to the blockchain.

6.6.1 Sealing and Unsealing Overhead

In Figure 6.7, we show overhead for sealing and the unsealing operation on the SGX platform. The x-axis represents the block size and the y-axis represents the CPU cost in milliseconds. By using a block size of 1,024 bytes, the average time it takes to seal a single batch record of 2.8 MB is 400 milliseconds compared to 2,000 milliseconds when using 32 bytes block size. With increasing block size, the time to seal and and unseal data reduces. This is as a result of reduction in frequency of number of blocks of data between the enclave and the untrusted module of the application.

6.7 Limitations and Future Work

In this work, we leveraged the immutability of the blockchain network to store audit information on how IoT data is stored and read by users. One of the main limitation of using blockchain is the scalability problem. This limitation is not pertinent to our solution, since the data is not always needed immediately and all pending writes and read can be processed and committed to the blockchain at a later time. In addition, this limitation does not apply to read operation as shown in the evaluation. One way to overcome this limitation is to use private blockchains. By using private blockchains, we can eliminate the time used to mine block since all participants in the network is permissioned or known.

6.8 Related Work

In this section, first, we provide discussion on related work on blockchain, and IoT system.

Blockchain With the increase in adoption of blockchain technology, various researchers have proposed different use cases for the new technology. Zyskind et al. (Zyskind et al., 2015) proposed using blockchain to decentralize storage of data. Our work differs from theirs, since we provide a full implementation that leveraged SGX secure computing to store raw data in order to defeat malicious attackers. Dorri et al. (Dorri et al., 2017) proposed using blockchain to manage IoT network. By providing a light weight blockchain consensus system, devices with low processing power can run blockchain independently. Various works exist on how to improve the performance of blockchain technology as seen in (Eyal et al., 2016).

IoT System and Applications In previous work by Earlence et al. (Fernandes et al., 2016), they show how a smart lock can be compromised by attacking the Samsung Smart Application. They demonstrated an attack that requested limited access permission to only

perform lock action on a smart lock, but instead gained full control privilege to also perform unlock action. Vijay et al. (Sivaraman et al., 2016) show how they capture non encrypted network traffic from Wemo device to perform a replay attack on the device.

6.9 Conclusion

As the adoption of IoT device usage increases, proper data access audit, data usage transparency, and data privacy are very critical due to the vast amount of data generated by these devices. This chapter introduces IoTSmartContract that offers a decentralized data access control policy system, data security, and data integrity by leveraging recent advances in blockchain technology and trusted computing using Intel SGX. Our approach utilizes the blockchain to manage data access to IoT data in a decentralized way and stores the raw encrypted data in SGX enabled platform by ensuring all data processing and storage is done in the secure enclaves. Our platform utilizes a real blockchain platform called Ethereum to evaluate our approach. We used a real Intel SGX platform to evaluate our secure storage platform.

CHAPTER 7

DISSERTATION SUMMARY

7.1 Cyberdeception based defenses

Effective evaluation of cyberdeceptive defenses is notoriously challenging. By leveraging both synthetic data generation and human subjects for running attacks, we show that evaluating deceptive defenses powered by deep learning is a promising approach. In particular, a combination of ensemble learning leveraging multiple classifier models, online adaptive metric learning, and novel class detection suffices to model aggressively adaptive adversaries who respond to deceptions in a variety of ways. Our case study evaluating an advanced, deceptive IDS shows that the resulting synthetic attacks can expose both strengths and weaknesses in modern embedded-deception defenses.

7.2 Domain Adaptation

In this work, we developed a framework that learns a mapping from low level data traces to high level concepts in terms of tactics deployed by the attacker. Due to the scarcity of labeled training data, we developed a novel domain adaptation technique that eliminates manual feature mapping methods on datasets generated from different domains and facilitates the use of these data across different domains.

7.3 Threat Report Classification

In this work, we explore the problem of classifying threat reports into tactics and technique categories that originate from various organizations due to the need to collaborate in defeating evolving attacks. To complete this work, we first leverage natural language processing techniques to perform threat report classification. Second, we apply bias correction meth-

ods to overcome bias in data from different sources. Lastly, we show that our approach outperforms TTPDrill with an increase in classifier accuracy by up to 78%.

7.4 Future Work

7.4.1 Cyberdeception based defenses

In this work, we focused mainly on the classification of attacks. Possible future direction is in the area of detection of attacks as multi-step activities. Future work could include representing the attack traces as a set of stream data where concept-drift could occur.

7.4.2 Domain adaptation

In future work, we will explore the prediction of kill-chain phase and techniques deployed by the attacker. We will also explore the segmentation of attack traces to determine how the attacker completed a sequence of actions using change detection techniques. In addition, we will build a hybrid system that can leverage data from more than two domains to train a classifier.

7.4.3 Threat report classification

Our current work focuses on classifying threat reports to different attack tactics and techniques. Future work in this direction could include extracting portions of the text that may be of interest to a security analyst. Another extension of this work could include the extraction of indicators of compromise that can be used to configure intrusion detection systems in an automated way.

APPENDIX

SMART CONTRACT DEFENSE THROUGH BYTECODE REWRITING [1]

A.1 Introduction

Recent increases in the adoption rate of smart contract applications have spurred initial coin offerings (ICOs)[2] and decentralized autonomous organizations (DAOs) to leverage multiple applications to raise money for disparate start-ups. This surge in investment has motivated a corresponding surge in smart contract attacks and vulnerability discoveries. For example, cybercriminals have leveraged re-entrancy attacks (Atzei et al., 2017; Konstantopoulos, 2018) and parity multisig wallet attacks (Breidenbach et al., 2017) to steal more than 60 million dollars in cryptocurrency.

As a result, various languages have been developed or modified to compile smart contracts as Ethereum bytecode. These include Solidity[3] (which resembles JavaScript), Haskell,[4] and Vyper.[5] Solidity is presently the most popular of these languages. However, developers are often reluctant to learn new languages, and gaining the proficiency to develop correct and secure code in a new language can be demanding.

These obstacles are exacerbated by the increasing complexity and subtlety of vulnerabilities leveraged by attackers to exploit and steal cryptocurrencies from blockchain networks. Various researchers have proposed automated tools for finding bugs in smart contracts before

[1] This chapter contains material previously published as: Gbadebo Ayoade, Erick Bauman, Latifur Khan, and Kevin W. Hamlen. "Smart Contract Defense Through Bytecode Rewriting." In *Proceedings of the Symposium on Recent Advances on Blockchain and its Applications*, July 2019. Ayoade led the design, implementation and evaluation of the approaches used in this research.

[2] https://www.icohotlist.com

[3] https://github.com/ethereum/solidity

[4] https://github.com/takenobu-hs/haskell-ethereum-assembly

[5] https://github.com/ethereum/vyper

deployment to the blockchain network. Most of these tools rely on the source code to carry out their analysis (Luu et al., 2016; Kalra et al., 2018), though a few (e.g., teEther (Krupp and Rossow, 2018)) perform bug-search at the bytecode level.

Rather than searching for bugs, our work leverages automated bytecode rewriting to allow developers to create smart contracts in any language, yet automatically enforce security policies at the bytecode level without relying on developer expertise to secure the application. Our framework ensures that vulnerable bytecode is properly protected without access to source code. By providing a framework that uses a source-agnostic approach, we can enforce security policy rules across different development tool chains.

Source-free, binary transformations are widely recognized as more difficult to implement than source-level analyses and transformations. Lack of contextual variable meanings (Wang et al., 2017), irregular instruction alignment of certain architectures (e.g., CISC native codes) (Bauman et al., 2018), and recovery of code control-flow graphs and function entry points (Wartell et al., 2014), are all perennial challenges documented in the literature. However, Ethereum bytecode has many syntactic properties that aid feasibility of binary rewriting of smart contracts relative to other binary languages, including strict instruction alignment and whitelisting of all indirect control-flow targets with `JUMPDEST` opcodes (Wood, 2019).

Bytecode rewriting of Ethereum contracts can therefore be achieved in four major steps: (1) Disassemble the bytecode to semantically equivalent assembly code. (2) Instrument the disassembled bytecode with new security guard code that enforces the desired policy. (3) Identify all jump locations and rewrite their destinations to match the code motions induced by the instrumentation step. (4) Verify that the modified code is *transparent* (Sridhar et al., 2014) with respect to the original code (i.e., it implements the same input-output relation whenever the security policy is not violated).

In this work, we build a framework that can rewrite Ethereum bytecode and update all jump instructions to reflect the new offset of their targets based on the modified code. Our

work differs from previous systems by creating a framework that can modify the Ethereum bytecode without the need of high level language source code (cf., (Lind et al., 2016; Luu et al., 2016)). In short, our contributions include the following (Ayoade et al., 2019).

- We propose and implement a framework to rewrite Ethereum bytecode without access to source code.
- Our framework detects vulnerable bytecode instructions and inserts guard code to mitigate attacker exploits.
- We implement Ethereum virtual machine code verification in the Coq theorem prover to machine-prove semantic transparency.
- We evaluate the system on real world smart contracts and measure the system overhead.

The rest of the chapter is organized as follows. Section A.2 provides background on smart contract and Ethereum bytecode vulnerabilities. Section A.3 discusses challenges and solutions encountered in designing a bytecode rewriter framework. Section A.4 provides the architecture of our system and Section A.5 describes our implementation. Section A.6 contains our evaluation, followed by discussion of related work in Section A.7. Finally, Section A.8 examines limitations and proposes future directions, and Section A.9 concludes.

A.2 Background

A.2.1 Ethereum Virtual Machine

Smart contracts are autonomous computations executed by decentralized entities on a blockchain. A popular smart contract framework implementation is the Ethereum virtual machine (EVM). The EVM is a stack based computer that executes a sequence of bytecode instructions. Its

state consists of a stack of 32-byte values, a memory, and a key-value store for persistent storage. EVM bytecode consists of more than 100 opcodes, such as `ADD`, `SUB`, `PUSH`, and `JUMP`. Each opcode has an associated fee called *gas* that must be paid to execute the instruction. Two token standards called ERC20 and ERC721 implement custom cryptocurrencies and custom non-fungible assets.

A.2.2 Common Ethereum Smart Contract Vulnerabilities

The increased use of smart contracts has resulted in the discovery of numerous contract vulnerabilities, including arithmetic over/underflow, smart contract owner hijacking, and re-entrancy attack vulnerabilities. We here focus on arithmetic vulnerability detection and mitigation. Integer overflows and underflows can occur during EVM code execution, leading to loss of tokens or money. The stack consists of up to 1024 32-byte words, each of which can hold a maximum value of 2^{256}. By adding a number to the max value, the new value rolls over to zero. Subtracting from a zero value dually rolls the result over to the maximum value[6] (ConsenSys, 2019) because EVM uses unsigned int256 types (Konstantopoulos, 2018). Integer underflow vulnerabilities can allow an attacker to roll over his initial balance to the maximum value, thereby gaining access to a large token balance that he does not own. This work focuses on mitigating these vulnerabilities by rewriting the smart contract bytecode.

A.3 Challenges

A.3.1 EVM Control-flows and Jump Retargeting

Since the EVM is a stack-based machine, EVM bytecode consists of a sequence of one-byte instructions (except for `PUSH` instructions, which contain immediate values). To control the flow of the program, the address of a jump destination is first pushed to the stack as an

[6]https://github.com/CoinCulture/evm-tools/blob/master/analysis/guide.md

input to the jump instruction, which is then executed. All jump destination addresses are marked with the `JUMPDEST` instruction. This is to ensure that programs can only jump to specific unique addresses marked in the bytecode. This mechanism is enforced by the EVM. To enforce this policy, the EVM parses the program bytecode and memorizes all the `JUMPDEST` targets. Every jump target is checked for validity before executing each `JUMP` instruction.

Unlike most native code architectures, where the machine code contains direct jump instructions, all jumps in EVM bytecode use the address at the top of the stack to identify the jump target. Code transformations that move instructions must therefore modify all jumps whose targets might have moved. We address this challenge in Section A.4.1.

A.3.2 Minimizing Overhead in Modified Bytecode

Protecting vulnerable code segments in the bytecode requires adding more instructions to the original bytecode. Inserting full guard code where vulnerable code exists results in a larger bytecode size, which affects the deployment cost on the Ethereum blockchain. For example, a smart contract with 100 bytes of code costs 2000 gas to deploy, while a smart contract with 50 bytes of code costs 1000 gas, resulting in a savings of 50%. According to the Ethereum yellow paper (Wood, 2019), every byte deployed on the blockchain costs 200 gas. As a result of this, we need an efficient technique to optimize the rewriting. We address this challenge in Section A.4.3.

A.3.3 Verifying Bytecode Correctness and Transparency

Bytecode rewriting is a potentially complex operation. To obtain high assurance, a machine-checked verification system allows us to verify that the modified bytecode program maintains the policy-compliant behaviors and correctness properties of the original code. To address this need, we build a verification tool that simulates the Ethereum stack VM and prove

Algorithm 5: EVM bytecode hardening

```
   Data: EVMbytecode, EVMGuardCode, VulnerableOpcode
   Result: HardenedEVMBytecode
1  while Inst ∈ Instructions do
2      Opcode := GetOpcode(Inst);
3      Operand := GetOperand(Inst);
4      if VulnerableOpcode = Opcode then
5          InsertCode(EVMGuardCode);
6  while Inst ∈ Instructions do
7      Opcode := GetOpcode(Inst);
8      Operand := GetOperand(Inst);
9      if Opcode = JUMP then
10         pushInst := GetPreviousPushInst();
11         oldTarget := getOperand(pushInst);
12         while Inst_2 ∈ Instructions do
13             if oldTarget = oldlabel(Inst_2) ∧ GetOpcode(Inst_2) = JUMPDEST then
14                 rewritePushInstruction(Inst, getnewlabel(Inst_2))
```

the transparency of the original and the modified bytecode. We address this challenge in Section A.4.4.

A.4 Architecture

As shown in Figure A.1, our system accepts policy rules and EVM bytecode as input. The framework consists of the Bytecode rewriter and the EVM code verifier. The bytecode rewriter consists of a disassembler, the rewriter, and an assembler. The bytecode rewriter output is fed to the EVM code verifier together with the original bytecode to ensure the rewritten program is equivalent. If the verifier succeeds, we output the hardened bytecode. If the verifier fails, we retry the rewriting step with a bounded time. In our system, we propose an in-lined bytecode insertion algorithm shown in Algorithm 5 and a function call technique shown in Algorithm 6.

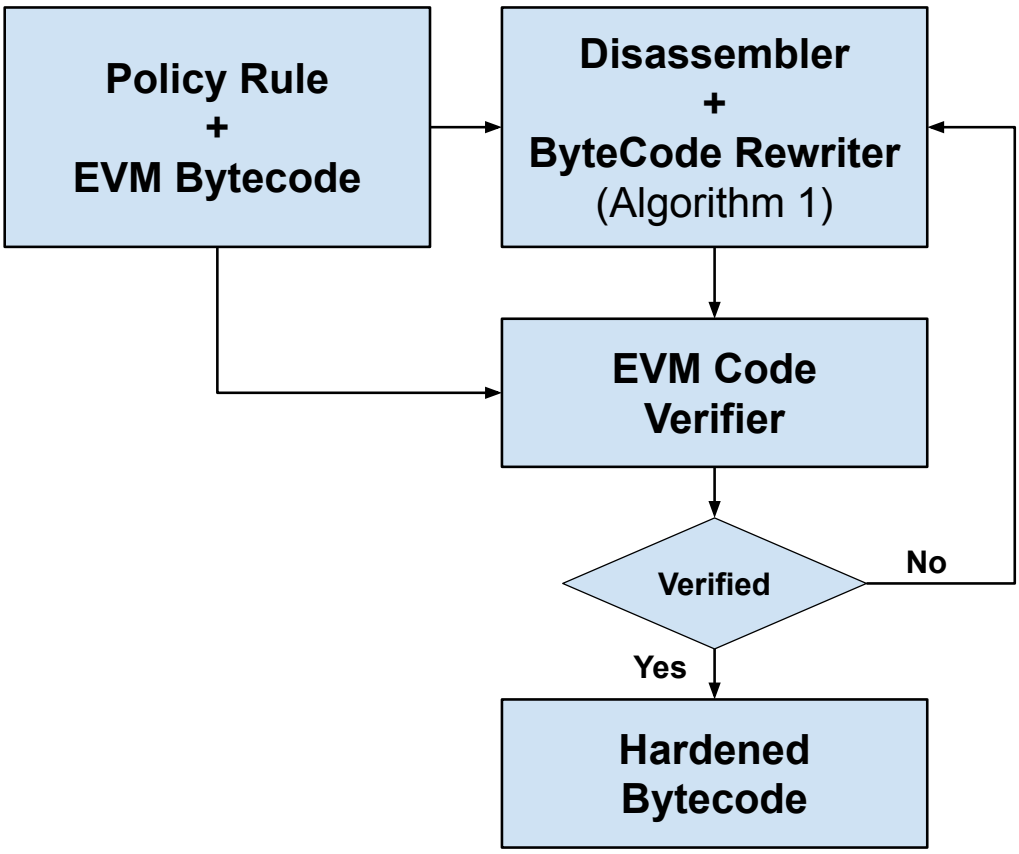


Figure A.1: System architecture

A.4.1 In-lined Bytecode Rewriter

In order to rewrite the EVM bytecode as shown in Algorithm 5, we first disassemble the bytecode to opcode instructions and extract the opcode and the operand of each instruction (lines 2–3). If the opcode is vulnerable, we insert the guard code before the vulnerable opcodes (line 5).

The resulting assembly code is misaligned due to the inserted code and `JUMP` instructions. To realign the code, we scan the code again for `JUMP` or `JUMPI` instructions (line 9). In most cases the instruction that precedes the `JUMP` opcode is a `PUSH` instruction, whereupon we extract the argument that specifies the jump target (line 11). We next scan the instructions

```
1  DUP1 // duplicate second subtraction argument
2  DUP3 // duplicate first subtraction argument
3  GT // test for underflow
4  NOT
5  PUSH [tag] n
6  JUMPI
7  REVERT // underflow detected
8  tag n
9  JUMPDEST
10 SUB // safely perform subtraction
```

Listing A.1: Underflow protection bytecode

to find a match between the extracted old jump target and the new jump target location label (line 12). After a match is found, we extract the offset of the new `JUMPDEST` instruction and rewrite the `PUSH` instruction's argument to the new jump target location (line 14).

A.4.2 Addressing the Policy Rule Generation Challenge

We must generate and convert a protection policy rule to bytecode, which can then be used as guard code to protect the vulnerable code. Since we perform our rewriting at the bytecode level, we can generate the guard bytecode once and apply it to any other bytecode compiled from any other language. In our case, we write our protection policy in Solidity and extract the compiled bytecode for the application.

For example, Listing A.1 shows the code generated for underflow code protection. The code checks whether the subtrahend exceeds the minuend before performing the subtraction to avoid a negative result, which is not supported by the EVM as discussed in Section A.2.2 (ConsenSys, 2019; Konstantopoulos, 2018). Program execution is aborted if an impending underflow is detected.

A.4.3 Optimized Guard Code Rewrite

Algorithm 6 addresses the challenge of optimizing the bytecode rewriting algorithm to minimize bytecode size and instruction count, as mentioned in Section A.3.2. In order to minimize

```
1 000000 PUSH ⟨current address⟩
2 000002 PUSH ⟨address of appended guard code⟩
3 000004 JUMP
```

Listing A.2: Function call code in EVM bytecode

the size of the binary file generated by inline guard code insertion, we utilized a function call-like system in the EVM bytecode. EVM does not support first-class function calls at the bytecode level.

To achieve our optimization, we inserted code that allows the program to remember how to return to the calling function after executing the guard code. In order to achieve this, function call code is first inserted before all vulnerable instruction code (line 5). Guard code is next appended to the current bytecode (line 6).

The function call code's `PUSH` argument is initialized with a placeholder location value that is later updated, the instruction is labeled as the current location, and the consecutive instruction is labeled as the function call instruction. To update the place holder location in the function call, we first scan the new code for the location of the appended guard code. Second, we scan the code for the labels; if the instruction label is the current location (line 9), we update the `PUSH` argument to the current location value to save the return address on the stack. Third, we scan the instructions for the function call `PUSH` instruction and we update the argument to the location of the appended guard code (lines 12). Finally, we rewrite the jump target locations using the steps from Algorithm 5.

Listing A.2 shows the code flow of the function call code routine. To call the appended guard code, the current address of the program instruction is first pushed to the stack. Second, the address of the guard code function is pushed to the stack and the jump instruction is executed to the jump to the guard code. After execution, the guard uses the saved location to return to the calling code position.

Algorithm 6: EVM bytecode optimized rewriter

```
Data: EVMbytecode, EVMGuardCode, Vulnerable_Opcode
Result: HardenedEVMBytecode
while Inst ∈ Instructions do
    Opcode := GetOpcode(Inst);
    Operand := GetOperand(Inst);
    if VulnerableOpcode = Opcode then
        InsertFunctionCallCode();
AppendCode(EVMGuardCode);
while Inst ∈ Instructions do
    instructionLabel := getInstructionLabel();
    if instructionLabel = saveCurrentInstAddress then
        UpdatePushInstrArg(Inst, currentLocation);
    if currInstructionLabel = functioncall then
        UpdatePushInstrArg(Inst, appendedCodeLocation);
Reuse steps 6–14 of Algorithm 5 to rewrite jump targets
```

$$
\begin{aligned}
&\rho : \mathbb{N} \rightharpoonup \iota && \text{(program)}\\
&\iota ::= \texttt{PUSH}\ n \mid \texttt{POP} \mid \texttt{SUB} \mid \texttt{JUMP} \mid \texttt{STOP} \mid \cdots && \text{(instruction)}\\
&\mu ::= \langle \sigma, \mathbf{m} \rangle \mid \langle pc, \sigma, \mathbf{m} \rangle && \text{(machine state)}\\
&\sigma ::= \cdot \mid n :: \sigma && \text{(stack)}\\
&\mathbf{m} : \mathbb{N} \rightharpoonup \mathbb{N} && \text{(memory)}
\end{aligned}
$$

$$
\frac{\rho(pc) = \texttt{PUSH}\ n}{\rho \vdash \langle pc, \sigma, \mathbf{m} \rangle \rightarrow_1 \langle pc+1, n :: \sigma, \mathbf{m} \rangle}(\text{PUSH})
$$

$$
\frac{\rho(pc) = \texttt{POP}}{\rho \vdash \langle pc, n :: \sigma, \mathbf{m} \rangle \rightarrow_1 \langle pc+1, \sigma, \mathbf{m} \rangle}(\text{POP})
$$

$$
\frac{\rho(pc) = \texttt{SUB} \qquad n_1 \geq n_2}{\rho \vdash \langle pc, n_1 :: n_2 :: \sigma, \mathbf{m} \rangle \rightarrow_1 \langle pc+1, (n_1 - n_2) :: \sigma, \mathbf{m} \rangle}(\text{SUB})
$$

$$
\frac{\rho(pc) = \texttt{JUMP} \qquad \rho(n) = \texttt{JUMPDEST}}{\rho \vdash \langle pc, n :: \sigma, \mathbf{m} \rangle \rightarrow_1 \langle n, \sigma, \mathbf{m} \rangle}(\text{JUMP})
$$

$$
\frac{\rho(pc) = \texttt{STOP}}{\rho \vdash \langle pc, \sigma, \mathbf{m} \rangle \rightarrow_1 \langle \sigma, \mathbf{m} \rangle}(\text{STOP})
$$

Figure A.2: EVM semantics (abbreviated and simplified)

A.4.4 EVM Code Verification

Here we address the challenge of bytecode verification as introduced in Section A.3.3. In order to verify the properties of the modified bytecode are still correct, we need a system

that allows us to specify theorems about program behaviors and prove their correctness. By leveraging the Coq interactive proof assistant, we implemented an EVM stack in Coq.

Figure A.2 shows a simplified and abbreviated definition of our EVM semantics for Coq. The semantics are formalized as a small-step machine in which bad states (e.g., stack underflows, invalid jumps, etc.) are intentionally left undefined. This makes unprovable any theorems that depend upon the EVM's behavior upon encountering such states. As a result, proved theorems guarantee that bad states are avoided.

A program ρ is formalized as a partial mapping from offsets to instructions ι. The program's current state includes the current instruction offset (program counter *pc*), the stack contents σ, and the memory contents $\mathbf{m}$. Each semantic rule executes one instruction by reading the opcode located at the current program counter offset and manipulating the stack and/or memory accordingly. Fall-through instructions increment the program counter, whereas jumps assign it a target offset. Programs that halt normally enter final state $\langle \sigma, \mathbf{m} \rangle$.

A.4.5 Proving Transparency

Proving transparency of our bytecode rewriter entails proving that all policy-adherent behaviors of the program are preserved after rewriting. For a given rewriter $R : \rho \to \rho$, the transparency theorem can be formalized as follows.

Theorem 1. *For all programs ρ, if $\rho \vdash \langle 0, \cdot, \mathbf{m} \rangle \to^* \langle \sigma', \mathbf{m}' \rangle$ is derivable, then $R(\rho) \vdash \langle 0, \cdot, \mathbf{m} \rangle \to^* \langle \sigma', \mathbf{m}' \rangle$ is derivable, where $\to^*$ is the reflexive, transitive closure of small-step relation $\to_1$.*

Proof. The theorem is proved by first generalizing the theorem statement's initial program counter for the original program (0) to an arbitrary offset *pc*, and rewriting the rewritten program's initial program counter 0 to $r(pc)$, where $r : \mathbb{N} \to \mathbb{N}$ is the mapping from old offsets to new (relocated) offsets implemented by R. Initial stack $\cdot$ is likewise generalized

to an arbitrary stack σ. This generalization of the theorem facilitates a natural number induction over the number of steps n in transitive relation $\rightarrow^*$. By case distinction, each small-step semantic rule in Figure A.2 yields a modified state that satisfies the theorem by inductive hypothesis. The rule for instruction `STOP` satisfies the base case of the induction, completing the proof. □

A.5 Implementation

We implemented our bytecode rewriter in Python. We utilized the Ethereum dataset of all smart contract bytecode stored on the Google big-query platform. We extracted a total of 155,175 unique smart contracts from a total of 2,195,890 smart contracts deployed on the Ethereum network. Of the 155,175 smart contracts, we extracted 64,033 ECR20 Ethereum smart contracts and 1,515 ECR721 Ethereum smart contracts. For each of the smart contract types, we instrumented 1,000 smart contracts with code protection for integer overflow and underflow for both addition and subtraction instructions. We executed our bytecode rewriter on an Intel Core i5 with 8GB of memory.

We implemented our EVM verification by extending a stack computer developed in Coq (Nardelli, 2011). As discussed in Section A.4.4, we implemented the following instructions in Coq: `PUSH`, `ADD`, `SUB`, `MULT`, `POP`, `DIV`, `LT`, `GT`, `DUP1`, `DUP2`, `SWAP1`, `SWAP2`, `EXP`, `ISZERO`, and `STOP`. This subset encompasses all instructions needed for our guard code implementations. Since all other EVM opcodes are preserved by our rewriter, their semantics are not needed in the proof of Theorem 1.

A.6 Evaluation

In this section, we discuss the overhead in terms of instruction counts, as shown in Figures A.3 and A.4. The x-axis lists the different types of Ethereum smart contract interfaces as ECR20, ECR720, and normal smart contracts. The y-axis records the overhead as the increase in

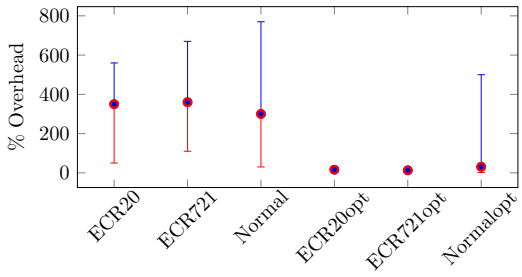


Figure A.3: MIN, AVG and Max instruction count overhead for integer overflow protection rewrite

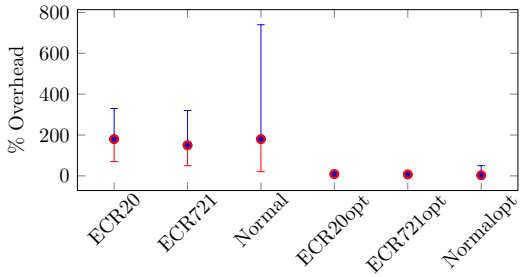


Figure A.4: MIN, AVG and Max instruction count overhead for integer underflow protection rewrite

Table A.1: Average instruction count overheads

Protection	**ECR20**	**ECR721**	**Norm**
Overflow	350%	360%	300%
Underflow	180%	150%	180%
Protection	**ECR20opt**	**ECR721opt**	**Normopt**
Overflow	16%	13%	31%
Underflow	9%	7%	3%

the instruction count of the modified code relative to the original code, giving the minimum, average, and maximum overhead for each. The non-optimized result represents the in-lined rewriting algorithm, and the optimized result represents the function call method.

For Figure A.3, the minimum instruction count percentage overhead for normal smart contracts is 30% for the non-optimized rewriter and 2% for the optimized rewriter. The average percentage overhead is 300% for non-optimized versus 31% for the optimized rewriter for normal smart contracts. Figure A.4 shows similar results. Table A.1 contains the average overheads for each type of contract.

To evaluate the overhead based on gas usage, we used the EVM simulator program developed by the Ethereum foundation to run a normal smart contract that is vulnerable to integer overflow and integer underflow, and we compared the gas usage to the smart contract protected by our rewriting framework. The execution overhead for the protected program from integer overflow and underflow is 300% for both. This result is due to similar instructions used to check if the parameters for addition or subtraction will not roll over to a zero or a maximum number as discussed in Section A.2.2.

A.7 Related Work

Smart Contract Code Defense Various researchers have explored finding vulnerabilities in smart contracts. Oyente (Luu et al., 2016) uses symbolic execution to run smart contracts and find predefined bugs such as re-entrancy attacks and insufficient balance. Osiris (Torres et al., 2018) extends Oyente to discover arithmetic vulnerabilities by using taint propagation techniques. Machine learning (Masud et al., 2008) based methods to classify code to detect vulnerable code have been explored. TeEther (Krupp and Rossow, 2018) allows automatic generation of exploits for smart contracts. Zeus (Kalra et al., 2018) leverages LLVM to generate LLVM IR code from an abstract syntax tree generated from solidity source code. The system needs access to the original source code to mitigate attacks.

Formal Verification Machine-verifying the correctness of security-sensitive programs for high assurance is becoming more important with the increase in security breaches. One main work in the area of program verification is compiler certification. Coq has been used to develop the first C compiler with an end-to-end, machine-checked proof of semantic transparency (Leroy, 2009). In order to verify the safety properties of smart contracts in Ethereum, an Ethereum smart contract verification system has been implemented in Isabelle/HOL (Hirai, 2017). Our work differs from these works by developing a framework

that provably mitigates smart contract vulnerabilities by inserting guard code in the raw bytecode.

A.8 Discussion and Future Work

In this work, we identified arithmetic vulnerabilities by searching for the occurrence of `ADD` and `SUB` instructions. Other instructions, such as the `SIGNEXTEND` opcode, can also be used to determine the bitwidth (Torres et al., 2018) and type inference of the result address. This instruction can help to determine whether an arithmetic overflow will occur. In addition, we identified jump target locations where the address of the `JUMP` instruction is pushed to the stack before the jump is executed.

For our formal verification, we focused mainly on verifying the operations of the Ethereum stack where most of the arithmetic operations occur. In future work, we will focus on adding the verification of memory access operations that can be useful in protecting against other Ethereum smart contract vulnerabilities.

For future work, we will identify other common jump operation patterns that involve function call patterns. In addition, we will use machine learning methods to detect vulnerabilities in smart contract bytecode.

A.9 Conclusion

This work explored bytecode rewriting as a mechanism for defending against smart contract vulnerabilities. Hardened EVM bytecode exhibited an average overhead of between 3% and 31% for both integer overflow and integer underflow guard code rewriting using our optimized bytecode rewriter. In addition, we implemented a code verification system within the Coq interactive theorem prover to machine-verify the transparency of the modified bytecode.

REFERENCES

Ahmed, M., A. N. Mahmood, and J. Hu (2016). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications 60*, 19–31.

Al-Khateeb, T., M. M. Masud, K. M. Al-Naami, S. E. Seker, A. M. Mustafa, L. Khan, Z. Trabelsi, C. Aggarwal, and J. Han (2016). Recurring and novel class detection using class-based ensemble for evolving data stream. *IEEE Transactions on Knowledge and Data Engineering 28*(10), 2752–2764.

Al-Naami, K., S. Chandra, A. Mustafa, L. Khan, Z. Lin, K. Hamlen, and B. Thuraisingham (2016). Adaptive encrypted traffic fingerprinting with bi-directional dependence. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ACSAC '16, New York, NY, USA, pp. 177–188. ACM.

Al-Naami, K., A. El Ghamry, M. S. Islam, L. Khan, B. M. Thuraisingham, K. W. Hamlen, M. Alrahmawy, and M. Rashad (2019). Bimorphing: A bi-directional bursting defense against website fingerprinting attacks. *IEEE Transactions on Dependable and Secure Computing*, 1–1.

Al-Shaer, E., J. Wei, K. W. Hamlen, and C. Wang (2019). *Autonomous Cyber Deception Reasoning, Adaptive Planning, and Evaluation of HoneyThings*. Springer.

Alnaami, K., G. Ayoade, A. Siddiqui, N. Ruozzi, L. Khan, and B. Thuraisingham (2015). P2V: Effective website fingerprinting using vector space representations. In *Proceedings of the IEEE Symposium on Computational Intelligence*, pp. 59–66.

Anagnostakis, K. G., S. Sidiroglou, P. Akritidis, M. Polychronakis, A. D. Keromytis, and E. P. Markatos (2010). Shadow honeypots. *International Journal of Computer and Network Security 2*(9), 1–15.

Anagnostakis, K. G., S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis (2005). Detecting targeted attacks using shadow honeypots. In *Proceedings of the USENIX Security Symposium*.

Applebaum, A., D. Miller, B. Strom, H. Foster, and C. Thomas (2017). Analysis of automated adversary emulation techniques. In *Proceedings of the Summer Simulation Multi-Conference*, SummerSim '17, San Diego, CA, USA, pp. 16:1–16:12. Society for Computer Simulation International.

Applebaum, A., D. Miller, B. Strom, C. Korban, and R. Wolf (2016). Intelligent, automated red team emulation. In *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ACSAC '16, New York, NY, USA, pp. 363–373. ACM.

Araujo, F., G. Ayoade, K. Al-Naami, Y. Gao, K. W. Hamlen, and L. Khan (2019, December). Improving intrusion detectors by crook-sourcing. In *Proc. Annual Computer Security Applications Conf.*.

Araujo, F. and K. W. Hamlen (2015). Compiler-instrumented, dynamic secret-redaction of legacy processes for attacker deception. In *Proceedings of the USENIX Security Symposium*.

Araujo, F., K. W. Hamlen, S. Biedermann, and S. Katzenbeisser (2014). From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 942–953.

Araujo, F., M. Shapouri, S. Pandey, and K. Hamlen (2015). Experiences with honey-patching in active cyber security education. In *8th Workshop on Cyber Security Experimentation and Test*.

Atzei, N., M. Bartoletti, and T. Cimoli (2017). A survey of attacks on Ethereum smart contracts SoK. In *Proc. Int. Conf. Principles of Security and Trust – Volume 10204*, pp. 164–186.

Avery, J. and E. H. Spafford (2017). Ghost patches: Fake patches for fake vulnerabilities. In *Proc. IFIP Int. Conf. ICT Systems Security and Privacy Protection*, pp. 399–412.

Awad, M., L. Khan, F. Bastani, and I.-L. Yen (2004). An effective support vector machines (SVMs) performance using hierarchical clustering. In *Proc. IEEE Int. Conf. Tools with Artificial Intelligence*, pp. 663–667.

Ayoade, G., F. Araujo, K. Al-Naami, A. M. Mustafa, Y. Gao, K. W. Hamlen, and L. Khan (2020, January). Automating cyberdeception evaluation with deep learning. In *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS)*, Grand Wailea, Maui.

Ayoade, G., E. Bauman, L. Khan, and K. W. Hamlen (2019, July). Smart contract defense through bytecode rewriting. In *Proceedings of the Symposium on Recent Advances on Blockchain and its Applications*, Atlanta, Georgia.

Ayoade, G., S. Chandra, L. Khan, K. Hamlen, and B. Thuraisingham (2018, Oct). Automated threat report classification over multi-source data. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pp. 236–245.

Ayoade, G., A. El-Ghamry, V. Karande, L. Khan, M. Alrahmawy, and M. Z. Rashad (2019, Aug). Secure data processing for iot middleware systems. *The Journal of Supercomputing 75*(8), 4684–4709.

Ayoade, G., V. Karande, L. Khan, and K. Hamlen (2018, July). Decentralized iot data management using blockchain and trusted execution environment. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pp. 15–22.

Bartos, K., M. Sofka, and V. Franc (2016). Optimized invariant representation of network traffic for detecting unseen malware variants. In *Proceedings of the USENIX Security Symposium*, Austin, TX, pp. 807–822.

Bauman, E., Z. Lin, and K. W. Hamlen (2018). Superset disassembly: Statically rewriting x86 binaries without heuristics. In *Proc. Annual Network & Distributed System Security Sym.*.

Baumrind, D. (1979). IRBs and social science research: The costs of deception. *IRB: Ethics & Human Research 1*(6), 1–4.

Ben-David, S., J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan (2010). A theory of learning from different domains. *Machine Learning 79*(1-2), 151–175.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning 2*(1), 1–127.

Benzel, T., R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab (2006). Experience with DETER: A testbed for security research. In *Proc. Int. Conf. Testbeds and Research Infrastructures for the Development of Networks and Communities*.

Bertino, E. (2016). Data privacy for iot systems: Concepts, approaches, and research directions. In *Big Data (Big Data), 2016 IEEE International Conference on*, pp. 3645–3647. IEEE.

Bhuyan, M. H., D. K. Bhattacharyya, and J. K. Kalita (2014). Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials 16*(1), 303–336.

Boggs, N., H. Zhao, S. Du, and S. J. Stolfo (2014). Synthetic data generation and defense in depth measurement of web applications. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, pp. 234–254.

Breen, C., L. Khan, and A. Ponnusamy (2002). Image classification using neural networks and ontologies. In *Proceedings. 13th International Workshop on Database and Expert Systems Applications*, pp. 98–102. IEEE.

Breidenbach, L., P. Daian, A. Juels, and E. G. Sirer (2017, July). An in-depth look at the Parity Multisig bug. Hacking, Distributed. http://hackingdistributed.com/2017/07/22/deep-dive-parity-bug.

Burger, E. W., M. D. Goodman, P. Kampanakis, and K. A. Zhu (2014). Taxonomy model for cyber threat intelligence information exchange technologies. In *Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security*, WISCS '14, New York, NY, USA, pp. 51–60. ACM.

Cabrera, J. B., L. Lewis, and R. K. Mehra (2001). Detection and classification of intrusions and faults using sequences of system calls. *ACM SIGMOD Record 30*(4), 25–34.

Canali, D., M. Cova, G. Vigna, and C. Kruegel (2011). Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the International Conference on World Wide Web*, pp. 197–206.

Chandola, V., A. Banerjee, and V. Kumar (2009). Anomaly detection: A survey. *ACM Computing Surveys 41*(3), 15.

Chechik, G., V. Sharma, U. Shalit, and S. Bengio (2010). Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research 11*, 1109–1135.

Chen, M., K. Q. Weinberger, F. Sha, and Y. Bengio (2014). Marginalized denoising auto-encoders for nonlinear representations. In *ICML*, pp. 1476–1484.

Cohen, W. W. (1995a). Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning*, pp. 115–123.

Cohen, W. W. (1995b). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, ICML'95, San Francisco, CA, USA, pp. 115–123. Morgan Kaufmann Publishers Inc.

ConsenSys (2019). Ethereum smart contract best practices: Known attacks. https://consensys.github.io/smart-contract-best-practices/known_attacks.

Costan, V. and S. Devadas (2016). Intel SGX explained. *IACR Cryptology ePrint Archive 2016*, 86.

Crane, S., P. Larsen, S. Brunthaler, and M. Franz (2013). Booby trapping software. In *Proc. New Security Paradigms Work.*, pp. 95–106.

Crosby, M., P. Pattanayak, and S. Verma (2016). Blockchain technology: Beyond bitcoin.

De Marneffe, M.-C. and C. D. Manning (2008). The stanford typed dependencies representation. In *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation*, pp. 1–8. Association for Computational Linguistics.

Dong, B., M. S. Islam, S. Chandra, L. Khan, and B. Thuraisingham (2018, Dec). Gci: A transfer learning approach for detecting cheats of computer game. In *2018 IEEE International Conference on Big Data (Big Data)*, pp. 1188–1197.

Dorri, A., S. S. Kanhere, and R. Jurdak (2017). Towards an optimized blockchain for iot. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, New York, NY, USA, pp. 173–178. ACM.

Dudorov, D., D. Stupples, and M. Newby (2013). Probability analysis of cyber attack paths against business and commercial enterprise systems. In *Proceedings of the IEEE European Intelligence and Security Informatics Conference*, pp. 38–44.

Dyer, K. P., S. E. Coull, T. Ristenpart, and T. Shrimpton (2012). Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the IEEE Symposium on Security & Privacy*, pp. 332–346.

Eskin, E., A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo (2002). A geometric framework for unsupervised anomaly detection. In *Applications of Data Mining in Computer Security*, pp. 77–101. Springer.

Eyal, I., A. E. Gencer, E. G. Sirer, and R. Van Renesse (2016). Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pp. 45–59.

Fernandes, E., J. Jung, and A. Prakash (2016, May). Security Analysis of Emerging Smart Home Applications. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*.

Filecoin (2017). Filecoin: A decentralized storage network. https://filecoin.io/filecoin.pdf. (Accessed on 08/09/2017).

Fireeye. Apt1 report. https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf. (Accessed on 12/12/2017).

Fireeye. Apt1 report. https://www.symantec.com/security-center/threats. (Accessed on 8/13/2018).

Forrest, S., S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff (1996). A sense of self for Unix processes. In *Proceedings of the IEEE Symposium on Security & Privacy*, pp. 120–128.

Foundation, E. (2014). Ethereum's white paper. https://github.com/ethereum/wiki/wiki/White-Paper. (Accessed on 08/09/2017).

Gao, Y., Y.-F. Li, S. Chandra, L. Khan, and B. Thuraisingham (2019). Towards self-adaptive metric learning on the fly. In *The World Wide Web Conference*, pp. 503–513. ACM.

Garcia-Teodoro, P., J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security 28*(1), 18–28.

Ghaith, H., A.-S. Ehab, A. Mohiuddin, C. Bei-Tseng, and N. Xi (2017). Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In *Annual Computer Security Applications Conference (ACSAC)*.

Gonzalez, N. M., W. A. Goya, R. de Fatima Pereira, K. Langona, E. A. Silva, T. C. M. de Brito Carvalho, C. C. Miers, J. E. Mångs, and A. Sefidcon (2016, Oct). Fog computing: Data analytics and cloud distributed processing on the network edges. In *2016 35th International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–9.

Greene, D. and P. Cunningham (2006). Practical solutions to the problem of diagonal dominance in kernel document clustering. In *Proceedings of the International Conference on Machine learning*, pp. 377–384. ACM.

Gubbi, J., R. Buyya, S. Marusic, and M. Palaniswami (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems 29*(7), 1645–1660.

Hirai, Y. (2017). Defining the Ethereum virtual machine for interactive theorem provers. In *Proc. Int. Conf. Financial Cryptography and Data Security*, pp. 520–535.

Hofmeyr, S. A., S. Forrest, and A. Somayaji (1998). Intrusion detection using sequences of system calls. *Journal of Computer Security 6*(3), 151–180.

Hu, H., G.-J. Ahn, and J. Jorgensen (2011). Detecting and resolving privacy conflicts for collaborative data sharing in online social networks. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, New York, NY, USA, pp. 103–112. ACM.

Huang, J., A. Gretton, K. M. Borgwardt, B. Schölkopf, and A. J. Smola (2007). Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pp. 601–608.

Hue, P. (2017). Philip hue iot portal. http://www2.meethue.com/en-us/. (Accessed on 08/09/2017).

Hutchins, E. M., M. J. Cloppert, and R. M. Amin (2011). Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research 1*, 80.

Jain, P., B. Kulis, I. S. Dhillon, and K. Grauman (2008). Online metric learning and fast similarity search. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pp. 761–768.

Jia, Y. J., Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, and A. Prakash (2017, February). ContexIoT: Towards Providing Contextual Integrity to Appified IoT Platforms. In *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS'17)*, San Diego, CA.

Jiang, J. and C. Zhai (2007). Instance weighting for domain adaptation in nlp. In *ACL*, Volume 7, pp. 264–271.

Jin, R., S. Wang, and Y. Zhou (2009). Regularized distance metric learning: Theory and algorithm. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pp. 862–870.

Juarez, M., S. Afroz, G. Acar, C. Diaz, and R. Greenstadt (2014). A critical evaluation of website fingerprinting attacks. In *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 263–274.

Kalra, S., S. Goel, M. Dhawan, and S. Sharma (2018). Zeus: Analyzing safety of smart contracts. In *Proc. Annual Network & Distributed System Security Sym.*.

Kanamori, T., S. Hido, and M. Sugiyama (2009). A least-squares approach to direct importance estimation. *The Journal of Machine Learning Research 10*, 1391–1445.

Kapravelos, A., Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna (2013). Revolver: An automated approach to the detection of evasive web-based malware. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, Washington, D.C., pp. 637–652. USENIX.

Karande, V., E. Bauman, Z. Lin, and L. Khan (2017). Sgx-log: Securing system logs with sgx. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, New York, NY, USA, pp. 19–30. ACM.

Kim, J., P. J. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, and J. Twycross (2007). Immune system approaches to intrusion detection—a review. *Natural Computing 6*(4), 413–466.

Konstantopoulos, G. (2018, January). How to secure your smart contracts: 6 solidity vulnerabilities and how to avoid them (part 1). *Loom Network J.*. https://bit.ly/2nNLuOr.

Kosba, A., A. Miller, E. Shi, Z. Wen, and C. Papamanthou (2016). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 839–858. IEEE.

Kreibichi, C. and J. Crowcroft (2004). Honeycomb – creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication Review 34*(1), 51–56.

Kruegel, C., D. Mutz, W. Robertson, and F. Valeur (2003). Bayesian event classification for intrusion detection. In *Proceedings of the Annual Computer Security Applications Conference*, pp. 14–23.

Kruegel, C. and G. Vigna (2003). Anomaly detection of web-based attacks. In *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 251–261. ACM.

Kruegel, C., G. Vigna, and W. Robertson (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks 48*(5), 717–738.

Krügel, C., T. Toth, and E. Kirda (2002). Service specific anomaly detection for network intrusion detection. In *Proceedings of the ACM Symposium on Applied Computing*, pp. 201–208.

Krupp, J. and C. Rossow (2018). teEther: Gnawing at Ethereum to automatically exploit smart contracts. In *Proc. USENIX Security Sym.*, pp. 1317–1333.

Lee, W. and D. Xiang (2001). Information-theoretic measures for anomaly detection. In *Proceedings of the IEEE Symposium on Security & Privacy*, pp. 130–143.

Leroy, X. (2009). Formal verification of a realistic compiler. *Communications ACM 52*(7), 107–115.

Li, W., Y. Gao, L. Wang, L. Zhou, J. Huo, and Y. Shi (2018). OPML: A one-pass closed-form solution for online metric learning. *Pattern Recognition 75*, 302–314.

Li, Y.-F., Y. Gao, G. Ayoade, H. Tao, L. Khan, and B. Thuraisingham (2019). Multistream classification for cyber threat data with heterogeneous feature space. In *The World Wide Web Conference*, WWW '19, New York, NY, USA, pp. 2992–2998. ACM.

Liao, X., K. Yuan, X. Wang, Z. Li, L. Xing, and R. Beyah (2016). Acing the ioc game: Toward automatic discovery and analysis of open-source cyber threat intelligence. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, New York, NY, USA, pp. 755–766. ACM.

Lind, J., I. Eyal, P. Pietzuch, and E. G. Sirer (2016). Teechan: Payment channels using trusted execution environments. *arXiv preprint arXiv:1612.07766*.

Linux Manual (2019). *editcap: Edit and/or Translate the Format of Capture Files*. https://linux.die.net/man/1/editcap.

Long, B., P. S. Yu, and Z. Zhang (2008). A general model for multiple view unsupervised learning. In *Proceedings of the 2008 SIAM international conference on data mining*, pp. 822–833. SIAM.

Luu, L., D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor (2016). Making smart contracts smarter. In *Proc. ACM Conf. Computer and Communications Security*, pp. 254–269.

Marceau, C. (2001). Characterizing the behavior of a program using multiple-length n-grams. In *Proceedings of the New Security Paradigms Workshop*, pp. 101–110.

Masud, M., J. Gao, L. Khan, J. Han, and B. M. Thuraisingham (2011, June). Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering 23*(6), 859–874.

Masud, M., L. Khan, and B. Thuraisingham (2011). *Data Mining Tools for Malware Detection*. CRC Press.

Masud, M. M., T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham (2008). Cloud-based malware detection for evolving data streams. *ACM Trans. Management Information Systems 2*(3).

Masud, M. M., T. M. Al-Khateeb, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham (2011). Detecting recurring and novel classes in concept-drifting data streams. In *Proceedings of the International IEEE Conference on Data Mining*, pp. 1176–1181.

Masud, M. M., J. Gao, L. Khan, J. Han, and B. Thuraisingham (2008). A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Proceedings of the International Conference on Data Mining*, pp. 929–934.

Masud, M. M., J. Gao, L. Khan, J. Han, and B. Thuraisingham (2010). Classification and novel class detection in data streams with active mining. In *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining*, pp. 311–324.

Masud, M. M., L. Khan, and B. Thuraisingham (2008, March). A scalable multi-level feature extraction technique to detect malicious executables. *Information Systems Frontiers 10*(1), 33–45.

MinIO (2019). Minio object storage. https://min.io/.

MITRE. Adversarial tactics, techniques and common knowledge. https://attack.mitre.org/wiki/Main_Page. (Accessed on 12/12/2017).

Mockaroo (2018). Product data set.

Mordor Intelligence (2018). Global cyber deception market. Technical report, Mordor Intelligence.

Mustafa, A. M., G. Ayoade, K. Al-Naami, L. Khan, K. W. Hamlen, B. Thuraisingham, and F. Araujo (2017, Dec). Unsupervised deep embedding for novel class detection over data stream. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1830–1839.

Nakamoto, S. (2009). A peer-to-peer electronic cash system. bitcoin.org. (Accessed on 08/09/2017).

Nardelli, F. Z. (2011). Modelling and verifying algorithms in Coq: An introduction. https://www.di.ens.fr/~zappa/teaching/coq/ecole11/summer/exercices/solutions/compiler-sol.v.

NextronSystems. Apt simulator. https://github.com/NextronSystems/APTSimulator. (Accessed on 06/06/2018).

OpenIOC. Open indicator of compromise. hhttps://www.fireeye.com/blog/threat-research/2013/10/openioc-basics.html. (Accessed on 12/12/2017).

Pan, S. J., J. T. Kwok, and Q. Yang (2008). Transfer learning via dimensionality reduction. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17*, pp. 677–682.

Pan, S. J. and Q. Yang (2010). A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on 22*(10), 1345–1359.

Panchenko, A., L. Niessen, A. Zinnen, and T. Engel (2011). Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the Annual ACM Workshop on Privacy in the Electronic Society*, pp. 103–114.

Patcha, A. and J.-M. Park (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks 51*(12), 3448–3470.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pp. 61–74. MIT Press.

Portnoff, R. S., S. Afroz, G. Durrett, J. K. Kummerfeld, T. Berg-Kirkpatrick, D. McCoy, K. Levchenko, and V. Paxson (2017). Tools for automated analysis of cybercriminal markets. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 657–666. International World Wide Web Conferences Steering Committee.

Portokalidis, G., A. Slowinska, and H. Bos (2006). Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. *ACM SIGOPS Operating Systems Review 40*(4), 15–27.

PyTorch (2019). Open source deep learning platform. https://pytorch.org/.

Quinlan, J. R. (1986, March). Induction of decision trees. *Mach. Learn. 1*(1), 81–106.

Rivest, R. L., A. Shamir, and L. Adleman (1978, February). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*(2), 120–126.

Sadowski, G. and R. Kau (2019, March). Improve your threat detection function with deception technologies. Technical Report G00382578, Gartner.

Santorini, B. (1990). Part-of-speech tagging guidelines for the penn treebank project (3rd revision). *Technical Reports (CIS)*, 570.

Santos, N., H. Raj, S. Saroiu, and A. Wolman (2014). Using arm trustzone to build a trusted language runtime for mobile applications. In *ACM SIGARCH Computer Architecture News*, Volume 42, pp. 67–80. ACM.

Selenium (2019). Selenium browser automation. http://www.seleniumhq.org.

Shi, X., Q. Liu, W. Fan, S. Y. Philip, and R. Zhu (2010). Transfer learning on heterogenous feature spaces via spectral transformation. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pp. 1049–1054. IEEE.

Shu, X., D. Yao, and N. Ramakrishnan (2015). Unearthing stealthy program attacks buried in extremely long execution paths. In *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 401–413.

Sivaraman, V., D. Chan, D. Earl, and R. Boreli (2016). Smart-phones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, New York, NY, USA, pp. 195–200. ACM.

slock (2017). Initial coin offering market. https://slock.it/. (Accessed on 08/09/2017).

Souders, S. (2007). *High Performance Web Sites: Essential Knowledge for Front-End Engineers*. O'Reilly.

Sridhar, M., R. Wartell, and K. W. Hamlen (2014, November). Hippocratic binary instrumentation: First do no harm. *Science of Computer Programming, Special Issue on Invariant Generation 93*(B), 110–124.

STIX. Structured threat information expression. https://oasis-open.github.io/cti-documentation. (Accessed on 12/12/2017).

Sugiyama, M., T. Suzuki, S. Nakajima, H. Kashima, P. von Bünau, and M. Kawanabe (2008). Direct importance estimation for covariate shift adaptation. *Annals of the Institute of Statistical Mathematics 60*(4), 699–746.

Sysdig (2019). Universal system visibility tool. https://github.com/draios/sysdig.

Tang, Y. and S. Chen (2005). Defending against internet worms: A signature-based approach. In *Proc. Annual Joint Conf. IEEE Computer and Communications Societies*, pp. 1384–1394.

TAXII. Trusted automated exchange of intelligence information. https://oasis-open.github.io/cti-documentation. (Accessed on 12/12/2017).

tcpdump (2019). Tcpdump and libpcap. https://www.tcpdump.org/.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints abs/1605.02688*.

Torres, C. F., J. Schütte, and R. State (2018). Osiris: Hunting for integer bugs in Ethereum smart contracts. In *Proc. Annual Computer Security Applications Conf.*, pp. 664–676.

Van Doorn, L. (2006). Hardware virtualization trends. In *ACM/Usenix International Conference On Virtual Execution Environments: Proceedings of the 2 nd international conference on Virtual execution environments*, Volume 14, pp. 45–45.

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM.

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research 11*(Dec), 3371–3408.

Wang, R., Y. Shoshitaishvili, A. Bianchi, A. Machiry, J. Grosen, P. Grosen, C. Krügel, and G. Vigna (2017). Ramblr: Making reassembly great again. In *Proc. Annual Network & Distributed System Security Sym.*.

Wang, T., X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg (2014). Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the USENIX Security Symposium*.

Warrender, C., S. Forrest, and B. Pearlmutter (1999). Detecting intrusions using system calls: Alternative data models. In *Proceedings of the IEEE Symposium on Security & Privacy*, pp. 133–145.

Wartell, R., Y. Zhou, K. W. Hamlen, and M. Kantarcioglu (2014). Shingled graph disassembly: Finding the undecidable path. In *Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining*, pp. 273–285.

Wei, Y., Y. Zhu, C. W.-k. Leung, Y. Song, and Q. Yang (2016). Instilling social to physical: Co-regularized heterogeneous transfer learning. In *AAAI*, pp. 1338–1344.

Williams, P. A. H. and V. McCauley (2016, Dec). Always connected: The security challenges of the healthcare internet of things. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 30–35.

Wood, G. (2019). Ethereum: A secure decentralised generalised transaction ledger. Technical Report 3e36772, Ethereum & Parity.

Wright, C. V., S. E. Coull, and F. Monrose (2009). Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the IEEE Network and Distributed Security Symposium*, pp. 237–250.

Wu, Y., B. L. Tseng, and J. R. Smith (2004). Ontology-based multi-classification learning for video concept detection. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, Volume 2, pp. 1003–1006. IEEE.

Xiang, S., F. Nie, and C. Zhang (2008). Learning a mahalanobis distance metric for data clustering and classification. *Pattern recognition 41*(12), 3600–3612.

Yamada, M., T. Suzuki, T. Kanamori, H. Hachiya, and M. Sugiyama (2011, June). Relative Density-Ratio Estimation for Robust Distribution Comparison. *ArXiv e-prints*.

Yu, Y.-l. and C. Szepesvári (2012). Analysis of kernel mean matching under covariate shift. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 607–614.

Zadrozny, B. (2004). Learning and evaluating classifiers under sample selection bias. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 114. ACM.

Zyskind, G., O. Nathan, et al. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pp. 180–184. IEEE.

BIOGRAPHICAL SKETCH

Gbadebo Ayoade began his career as a software engineer in 2009 at one of the top electronic financial transaction companies serving the largest banks in Africa. In 2013, he decided to pursue further studies to further his knowledge in the area of reliable and secure computing. In 2014, he graduated from The University of Texas at Dallas with an MS in Computer Science. In 2015, he got a full scholarship to pursue his PhD program in Computer Science at The University of Texas at Dallas under the supervision of Dr. Latifur Khan and Dr. Kevin W. Hamlen. During his PhD studies, he was a member of the Big Data Analytics and Management Lab as a Research Assistant. His research interest ranges from big data systems, cyber security and applied machine learning. He also enjoys developing software for production systems.

CURRICULUM VITAE

Gbadebo Gbadero Ayoade

December 15, 2019

Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080-3021, U.S.A.

Email: gbadebo.ayoade@utdallas.edu

Educational History:

BS, Computer Science and Engineering, Obafemi Awolowo University, 2008
MS, Computer Science, University of Texas At Dallas, 2014
PhD, Computer Science, University of Texas At Dallas, 2019

Mitigating Cyberattacks with Machine Learning-based Feature Space Transforms
PhD Dissertation
Computer Science Department, The University of Texas At Dallas
Advisors: Dr. Latifur Khan and Dr. Kevin W. Hamlen

Employment History:

Research Assistant, The University of Texas at Dallas, September 2014 – present
Data Science Intern, Procter and Gamble, May 2019 – August 2019
Big Data Intern, Verizon Labs, May 2016 – August 2016
Software Engineering Intern, Fedex, May 2014 – August 2014
Software Engineer, Interswitch, Sept 2009 – January 2013

Professional Recognitions and Honors:

Erik Jonsson School of Engineering and Computer Science Endowed Scholarship Fund, UTD, 2014