Reasoning about Cyber Threat Actors

by

Eric Francis Nunes

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved May 2018 by the
Graduate Supervisory Committee:

Paulo Shakarian, Chair
Gail-Joon Ahn
Chitta Baral
Nancy Cooke

ARIZONA STATE UNIVERSITY

August 2018

ABSTRACT

Reasoning about the activities of cyber threat actors is critical to defend against cyber attacks. However, this task is difficult for a variety of reasons. In simple terms, it is difficult to determine *who* the attacker is, *what* the desired goals are of the attacker, and *how* they will carry out their attacks. These three questions essentially entail understanding the attacker's use of deception, the capabilities available, and the intent of launching the attack. These three issues are highly inter-related. If an adversary can hide their intent, they can better deceive a defender. If an adversary's capabilities are not well understood, then determining what their goals are becomes difficult as the defender is uncertain if they have the necessary tools to accomplish them. However, the understanding of these aspects are also mutually supportive. If we have a clear picture of capabilities, intent can better be deciphered. If we understand intent and capabilities, a defender may be able to see through deception schemes.

In this dissertation, I present three pieces of work to tackle these questions to obtain a better understanding of cyber threats. First, we introduce a new reasoning framework to address deception. We evaluate the framework by building a dataset from DEFCON capture-the-flag exercise to identify the person or group responsible for a cyber attack. We demonstrate that the framework not only handles cases of deception but also provides transparent decision making in identifying the threat actor. The second task uses a cognitive learning model to determine the intent – goals of the threat actor on the target system. The third task looks at understanding the capabilities of threat actors to target systems by identifying at-risk systems from hacker discussions on darkweb websites. To achieve this task we gather discussions from more than 300 darkweb websites relating to malicious hacking.

ACKNOWLEDGMENTS

I would like to thank my Ph.D advisor, Professor Paulo Shakarian, for his guidance throughout the dissertation. I am truly fortunate to have had the opportunity to work with him. I am grateful for his scientific knowledge, methodical advice, and many keen suggestions and discussions. He also encouraged me to collaborate with right set of people to work on my research problems. In addition, I would like to thank my defense committee members Prof. Gail-Joon Ahn, Prof. Chitta Baral, and Prof. Nancy Cooke.

I would also like to thank Dr. Gerardo I. Simari with whom I have collaborated extensively during the later half of my PhD. I have also had the pleasure of collaborating with multiple researchers during my PhD. I would like to thank Dr. Christian Lebiere, Dr. Robert Thomson, Dr. Holger Jaenisch, Stefano Bennati, Andrew Ruef, Jay Little, Jana Shakarian, Andrew Gunn, Casey Buto and Amanda Thart.

I have enjoyed the company and support of my friends and collaborators from CySIS lab: Elham, Ashkan, Ruocheng, Ericsson, Hamid, Soumajyoti, Mohammed, Ahmad, Vivin, Abhinav, John, Vineet, Nimish over the course of my program.

I especially thank my mom, dad, and sister. In simple words, I would not have made it this far without their support. I know that I always have them to count on.

This dissertation would not have been possible without funding from the U.S. Department of the Navy, Office of Naval Research, grant N00014-15-1-2742 and NEPTUNE program, the Arizona State University (ASU) Global Security Initiative (GSI), ASU Institute for Social Science Research (ISSR), CONICET and Universidad Nacional del Sur, Argentina and the EU H2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement 690974 for the project “MIREL”. I would also like to thank Cyber Reconnaissance, Inc. for providing the cyber threat intelligence data. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the funding agencies.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

1.1 Reasoning about threat actors

Reasoning about the activities of cyber threat actors is critical to defend against cyber attacks. In doing so, three broad factors need to be considered: the attacker's use of deception, the capabilities available, and the intent of launching the attack (see Fig. 1.1). For a threat actor to pose risk, he/she must intend to do harm and have the capabilities and resources to conduct a cyber attack [120, 81, 96, 127]. If we understand intent and capabilities, a defender may be able to see through deception schemes to identify the attacker – termed as cyber attribution. For real-world applications, security vendors take into consideration a threat actor's ability to pose a threat while designing products and services for threat assessment [40, 39]. The analysis that underpins reasoning about the activities of threat actors involves many diverse sources of data: network traffic, malware analysis including code similarity, and intelligence analysis from various hacker forums, to name a few. Independent and diverse sources of reporting strengthen an analytic argument. The process, flaws, outcomes, and methodology of understanding and identifying such threat actors have become a subject of increasingly broad interest over the past few years. Part of this is due to the increase in cyber-activity and the intersection of that cyber-activity with the public sphere. For example, it used to be that a major company being hacked would be of concern only to that company and its customers; however, the compromise of Sony Pictures allegedly by North Korea in late 2014 elevated public interest in the accurate attribution of cyber-aggression to the national level.

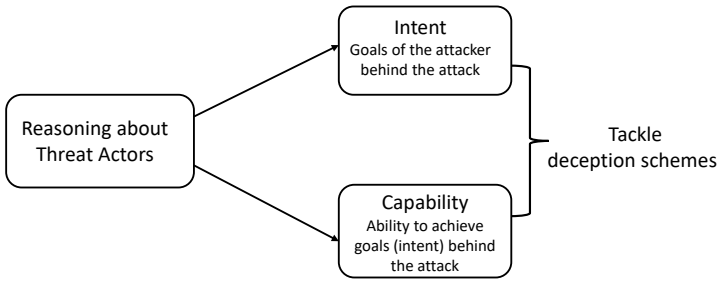


Fig. 1.1: Reasoning about threat actors.

1.2 Literature Overview

Previous research on understanding cyber threat landscape has focused on vulnerability analysis and prediction using public vulnerability disclosures databases like NVD [138, 16] and Twitter [106] as data sources. The reported results show poor predictive capability of NVD [86] and that the data sampling is not reflective of real world scenarios [21] but none leverage darkweb [1] sites as a source of intelligence for assessing threats. We observe that darkweb marketplaces gives threat actors access to resources that can be leveraged to carry out cyber attacks. Determining the intentions of a threat actor behind an attack is important to assess a threat. Intent refers to goals the attacker is trying to achieve on the compromised system. This can be inferred by analyzing malware/exploits used in the attack. Most malware analysis techniques look at identifying if the software is malicious or not [38, 123] or identifying which family the malware belongs to [8, 60, 61] without identifying the intentions of the software which in practice is largely human driven. More recently, there has been work on directly inferring the intent of a malware [54]. This approach leverages static malware analysis (i.e. analysis of the malware sample conducted without execution, such as decompilation) and a comparison with a crowd-source database of code snippets using a proprietary machine leaning approach. However, a key shortcoming of the static

[1] "darkweb" refers to the anonymous communication provided by crypto-networks like "Tor"

method is that it is of limited value when the malware authors encrypt part of their code as we saw with the infamous Gauss malware [59]. The goal of understanding a threat can lead to better cyber attribution. Identifying the threat actors responsible for a given cyber event is challenging due to lack of a reasoning framework to handle deception and representative data to evaluate proposed frameworks in the literature. Currently cyber attribution looks to identify machines, as opposed to a given hacker and his/her affiliations [14] or clustering common IP sources together [34].

1.3 Motivation

Most technologies designed to aid in cyber security are "introspective" meaning that they focus on examining vulnerabilities and abnormalities of the system being defended. Tools for this type of analysis have ranged from the traditional firewall and intrusion detection system to more recent systems using sophisticated anomaly detection or threat signatures using information shared amongst different organizations. However, an alternative viewpoint is to examine the environment of malicious cyber threat actors that lets an organization get a better understanding of the threat landscape in terms of intent and capability of the threat actor to defend itself and also identify the threat actor responsible for conducting the cyber attack.

In the spring of 2017, the Internet was gripped by a widespread ransomware attack, dubbed WannaCry; the ransomware spread as a worm affecting 300,000 machines in 150 countries, holding files hostage with encryption and promising decryption if a payment was made to a Bitcoin address. Hackers took advantage of the fact that many systems were not updated with the patch released by Microsoft, leaving them vulnerable. The attack was in fact discussed on darkweb forums in several languages including English and Russian as identified by cybersecurity company CYR3CON [122]; they also reported that hackers choose medical institutions as prime targets based on the history of paid ransom from

similar institutions. Hence the discussions on these darkweb platforms provides valuable information regarding the capabilities as well as the intent of threat actors and should be considered in identifying them.

Why do we reason about the activities of threat actors, and who is the customer of this reasoning? Law enforcement and the courts care about cyber attribution decisions when making investigative or legal decisions taking into account the intent and capability of the threat actor. In other spheres, it can help guide the direction and proportion of an organizational response to a cyber attack. For example, if a commercial company can determine if an attacker is part of an unsophisticated hacktivist gang rather than a sophisticated criminal enterprise (based on the intent and capability of the group), they could simply re-install the compromised computers as a defensive response rather than engaging with law enforcement. Likewise, according to Wheeler et al., "many offensive techniques, such as computer network attack, legal action (e.g., arrests and lawsuits), and kinetic energy attacks, can only be deployed if the source of the attack can be attributed with high confidence" [136].

In addition to understanding and identifying threat actors, we also ask: how do researchers train and evaluate these models? Using cyber attack data gathered from the real world is problematic for a few reasons. First, it is difficult to get real-world data due to the sensitive nature of the data. Additionally, even if the data were available, it is difficult to trust ground truth about that data. Could attackers' deceptions go unnoticed in this data? Who can say? To enable researchers to develop and evaluate their tools, we provide pre-processed data from capture-the-flag (CTF) contests, where access to the ground truth is available.

1.4 Outline of the Dissertation and Contributions

The central goal of this dissertation is to understand and reason about the activities of threat actors in terms of the intent and capability to conduct a cyber attack with the goal of

performing cyber attribution. The dissertation is divided in three parts. In the first part, we introduce a new reasoning framework to address deception. We evaluate the framework by building a dataset from DEFCON capture-the-flag exercise to identify the person or group responsible for a cyber attack. We demonstrate that the framework not only handles cases of deception but also provides transparent decision making in identifying the threat actor (Chapter 2 and 3). In the second part, we introduce a cognitive learning model to determine the intent – goals of the threat actor on the target system by analyzing the malware/exploits used (Chapter 4). The third part looks at understanding the capabilities of threat actors. To achieve this task we first gather discussions from more than 300 darkweb websites relating to malicious hacking (Chapter 5). Then, we leverage the reasoning framework to identify at-risk target systems (capable of exploitation) from hacker discussions on darkweb websites (Chapter 6). Finally, the dissertation is concluded in Chapter 7 with a summary of results presented in this work and future research directions. Literature corresponding to each of the specific problem is reviewed in the corresponding Chapter.

Chapter 2: Cyber-Deception and Attribution in Capture-the-Flag Exercises

Attributing the culprit of a cyber-attack is widely considered one of the major technical and policy challenges of cyber-security. The lack of ground truth for an individual responsible for a given attack has limited previous studies. In this Chapter, we overcome this limitation by leveraging DEFCON capture-the-flag (CTF) exercise data where the actual ground-truth is know. We use various classification techniques to identify the culprit in a cyberattack and find that deceptive activities account for the majority of misclassified samples. We also explore several heuristics to alleviate some of the misclassification caused by deception.

Chapter 3: Argumentation Models for Cyber Attribution

In this Chapter, we employ a more principled approach to counter deception based on the previously established theoretical framework for reasoning about cyber-attribution [114].

In particular we employ temporal reasoning to tackle the problem of deceptive attacks encountered in Chapter 2. We tackled an interesting research problem of identifying hacking group from a series of attacks over a period of time – differentiating between deceptive hacking groups.

We introduce an argumentation model based on the DeLP (Defeasible Logic Programming) framework designed to aid analysts in attributing cyber-attacks. Apart from the basic argumentation machinery, the framework makes use of latent variables to reduce the space of possible culprits (attackers)—the resulting system is therefore a hybrid between classical knowledge representation and reasoning techniques and machine learning classifiers. We report on results obtained from a prototype implementation, showing that our approach yields much higher accuracy than approaches reported in Chapter 2 (evaluated on the same dataset) that rely on machine learning classifiers alone—a jump from 37% to 64.5%. The result is an efficient and scalable reasoning framework designed to aid analysts in attributing cyber-attacks.

Chapter 4: Determination of Adversarial Intent

Malware reverse-engineering, specifically, identifying the tasks (intent) a given piece of malware was designed to perform (e.g., logging keystrokes, recording video, establishing remote access) is a largely human-driven process that is a difficult and time-consuming operation. In this chapter, we present an automated method to identify malware tasks using two different approaches based on the ACT-R cognitive architecture, a popular implementation of a unified theory of cognition. Using three different malware collections, we explore various evaluations for each of an instance-based and rule-based model - including cases in which the training data differs significantly from test; where the malware being evaluated employs packing to thwart analytical techniques; and conditions with sparse training data. We find that our approach based on cognitive inference consistently out-performs the current state-of-the art software for malware task identification as well as standard machine learning

approaches - often achieving an unbiased F1 score of over 0.9.

Chapter 5: Darkweb and Deepweb Mining for Cybersecurity Threat Intelligence

In this Chapter, we present an operational system for cyber threat intelligence gathering from various social platforms on the Internet particularly sites on the darkweb and deepweb. We focus our attention to collecting information from hacker forum discussions and marketplaces offering products and services focusing on malicious hacking. We have developed an operational system for obtaining information from these sites for the purposes of identifying emerging cyber threats. At the time of development the system was actively collecting approximately 305 cyber threats each week. These threat warnings include information on newly developed malware and exploits that have not yet been deployed in a cyber-attack. This provides a significant service to cyber-defenders. The system is significantly augmented through the use of various data mining and machine learning techniques. With the use of machine learning models, we are able to recall 92% of products in marketplaces and 80% of discussions on forums relating to malicious hacking with high precision. We provide analysis on the data collected, demonstrating its application to aid a security expert for better threat analysis.

Chapter 6: At-Risk System Identification via Analysis of Discussions on the Darkweb

Threat assessment of systems is critical to organizations' security policy. Identifying systems likely to be at-risk by threat actors can help organizations better defend against likely cyber attacks. Currently, identifying such systems to a large extent is guided by the Common Vulnerability Scoring System (CVSS). Previous research has demonstrated poor correlation between a high CVSS score and at-risk systems. In this Chapter, we leverage hacker discussions on darkweb marketplaces and forums collected using the system introduced in Chapter 5 to identify the platforms, vendors, and products likely to be at-risk by hackers. This gives us an indicator regarding the hacker capability of targeting systems based on their

discussions.

We employ and modify the reasoning system introduced in Chapter 3 that combines DeLP (Defeasible Logic Programming) and machine learning classifiers to identify systems based on hacker discussions observed on the darkweb. The modified system takes into account the hierarchical structure of identifying a system in terms of its platform, vendor and product. The system is evaluated on hacker discussions from nearly 300 darkweb forums and marketplaces. We improved precision by 15%–57% while maintaining recall over baseline approaches.

Chapter 7: Conclusion and Future Work

In this chapter, we first recapitulate the main ideas and results presented in the dissertation. Then some directions for extending the dissertation are given.

1.5 Summary of Contributions

Table 1.1 summarizes the contributions of the dissertation. The contributions can be split into three broad parts: 1) construction of reasoning framework capable of handling inconsistent and contradictory information in case of deception and to identify at-risk systems 2) construction of models for automated malware analysis to determine adversarial intent – the goals of the attacker on the host system and 3) Collected and pre-processed datasets to evaluate cyber attribution models and applications relating to cyber threat intelligence.

Table 1.1: Contribution Summary

Contribution	**Summary**
Reasoning framework [92, 94, 93]	• A framework that combines classical knowledge representation and reasoning techniques (DeLP) and machine learning classifiers to handle inconsistent and contradictory information. • The framework allows for transparent devision making aiding a security analyst to review the decision made. • Applied to the problem of cyber-attribution with improved performance over leveraging only machine learning classifiers. • Applied to the problem of At-risk system identification from hacker discussions on darkweb by modifying the framework to handle hierarchical decision making.
Cognitive Models [87, 124, 63]	• Two different models based on the ACT-R cognitive architecture - Instance-based and Rule-based for automated malware analysis to determine the goals of the attacker on the host system. • Evaluated on three different malware collections (with encrypted and mutated malware samples) with parameter exploration and scalability experiments.
Data Collection [92, 89, 103]	• We assemble and make available a dataset of cyber-attacks with ground truth derived from the traffic of DEFCON CTF. • Hacker discussions and items gathered from more than 300 darkweb forums and marketplaces in an automated way in real time.

Chapter 2

CYBER-DECEPTION AND ATTRIBUTION IN CAPTURE-THE-FLAG EXERCISES

2.1 Introduction

Attributing the culprit of a cyber-attack is widely considered one of the major technical and policy challenges of cyber-security. The lack of ground truth for an individual responsible for a given attack has limited previous studies. In this work, we take an important first step toward developing computational techniques toward attributing the actual culprit (here hacking group) responsible for a given cyber-attack. We leverage DEFCON capture-the-flag (CTF) exercise data which we have processed to be amenable to various machine learning approaches. Here, we use various classification techniques to identify the culprit in a cyber-attack and find that deceptive activities account for the majority of misclassified samples. We also explore several heuristics to alleviate some of the misclassification caused by deception. In this chapter:

- We assemble a dataset of cyber-attacks with ground truth derived from the traffic of the CTF held at DEFCON 21 in 2013.
- We analyze this dataset to identify cyber-attacks where deception occurred.
- We frame cyber-attribution as a multi-class classification problem and leverage several machine learning approaches. We find that deceptive incidents account for the vast majority of misclassified samples.
- We introduce several pruning techniques and show that they can reduce the effect of deception as well as provide insight into the conditions in which deception was employed by the participants of the CTF.

2.2 Dataset

2.2.1 Background

The DEFCON security conference sponsors and hosts a capture the flag (CTF) competition every year, held on site with the conference in Las Vegas, Nevada. DEFCON CTF is one of the oldest and best-known competitions. The ctftime.org site provides a ranking for CTF teams and CTF competitions, assigning it the highest average weight.

CTF competitions can be categorized by what role the competitors play in the competition: either red team, blue team, or a combination. In a *blue team* focused CTF, the competitors harden their systems against a red team played by the organizers of the CTF. In a *combined red/blue team* CTF, every team plays both blue and red team simultaneously. The NCCDC and CDX competitions are examples of a blue team CTF, while DEFCON CTF is a combined red/blue team. Each team is simultaneously responsible for hardening and defending their systems as well as identifying vulnerabilities and exploiting them in other teams' systems.

The game environment is created primarily by the DEFCON CTF organizers. The game focuses around programs (known in the game as *services*) written by the organizers and engineered to contain specific vulnerabilities. The binary images of the services are made available to each team at the start of the game, but no other information is released. Part of the challenge is identifying the purpose of each service, as well as the vulnerabilities they present. Identification of vulnerabilities serves both a defensive and offensive goal; once a vulnerability has been identified, a team may patch this vulnerability in the binary program. Additionally, the teams may create *exploits* for that vulnerability and use them to attack other teams and capture digital flags from those teams' systems.

Each team is also provided with a *server* running the services, which contains the digital flags to be defended. To deter defensive actions such as powering off the server or stopping

the services, the *white team* (a third team, played by the organizers) conducts periodic availability tests of the services running on each team's server. A team's score is the sum of the value of the flags they have captured, minus the sum of the flags that have been captured from that team, multiplied by an availability score determined by how often the white team was able to test that team's services. This scoring model incentivizes teams to keep their server online, identify the vulnerabilities in services and patch them quickly, and exploit other teams' services to capture their flags. It disincentivizes teams from performing host-level blocking and shutting down services, as this massively impacts their final score.

This game environment can be viewed as a microcosm of the global Internet, and the careful game of "cat and mouse" between hacking groups and companies. Teams are free to use different technical means to discover vulnerabilities—they may use fuzzing and reverse engineering on their own programs, or they may monitor the network data sent to their services and dynamically study the effects that network data has on unpatched services. If a team discovers a vulnerability and uses it against another team, the first team may discover that their exploit is re-purposed and used against them within minutes.

The organizers of DEFCON CTF capture all of the network traffic sent and received by each team, and publish this traffic at the end of the competition [35]. This includes IP addresses for source and destination, as well as the full data sent and received and the time the data was sent or received. This data is not available to contestants; depending on the organizers' choice from year to year, the contestants either have a real-time feed but with the IP address obscured, or a full feed delivered on a time delay of minutes to hours.

2.2.2 Analysis

We use the data from the CTF tournament held at DEFCON 21 in 2013; the dataset is very large, about 170 GB in compressed format. We used multiple systems with distributed and coordinated processing to analyze the data—fortunately, analyzing individual streams is

Table 2.1: Fields in an instance of network attack

Field	**Intuition**	**Example Value**
byte_hist	Histogram of byte sequences in the payload	0×43:245, 0×69:8, $0\times3a$:9, ...
inst_hist	Histogram of instructions used in the payload	cmp:12, subs:8, movtmi:60 .....
from_team	The team where the payload originates (attacking team)	Blue Lotus
to_team	The team being attacked by the exploit	Robot Mafia
time	Indicates the date and time of the attack	2013-08-03T23:45:17

easy to parallelize. We identified the TCP ports associated with each vulnerable service and then used the open source tool *tcpflow* to process the network captures into a set of files, with each file representing data sent or received on a particular connection.

With these data files identified, we analyzed some of them by hand using the Interactive Disassembler (IDA) to determine if the data contained shell-code, which in fact was the case. We used an automated tool to produce a summary of each data file as a JSON encoded element. Included in this summary was a hash of the contents of the file and a histogram of the processor instructions contained in the file. These JSON files were the final output of the low-level analysis, transforming hundreds of gigabytes of network traffic into a manageable set of facts about exploit traffic in the data. Each JSON file is a list of tuples (time-stamp, byte-histogram, instruction-histogram, attack team and target team). The individual fields of the tuple are listed in Table 2.1. This pre-processing phase can be summarized in the following steps:

- Un-tar the archives available from the organizers; this produces a large number of pcap-ng formatted files that contain the traffic captures.

- Convert the pcap-ng files to tcpdump format capture using the editcap utility. This will allow tcpflow to process the data.

- Use xargs and GNU parallel to run tcpflow on each pcap. This took some time, and produced a directory structure with files for data sent and received on host-port socket pairs. This step of processing allows file-based tools to process the network data.

- Finally, develop (and run, in parallel) a tool to process each file containing data sent or received by network ports associated with CTF challenges. This produced summary statistics for each data stream: a byte histogram, overall size, a hash, and an ARM instruction histogram (we ran a linear sweep with the Capstone instruction decoder to produce this). This data was saved via JSON.

After this pre-processing of the network data packets, we have around 10 million network attacks consisting of about 1 million unique exploits built and used by 20 teams in the competition. In order to attribute an attack to a particular team, apart from analyzing the payloads used by the team, we also need to analyze the behavior of the attacking team towards their adversary. For this purpose, we separate the network attacks according to the team being targeted. Thus, we have 20 such subsets (Table 4.3), which we denote with T-i, where $i \in \{1, 2, 3, ..., 20\}$. We also define deceptive attacks where multiple teams target the same team using the same exploit. Fig. 2.1 shows the distribution of unique deceptive attacks with respect to the total unique attacks in the dataset based on the target team. These unique deceptive attacks amount to just under 35% of the total unique attacks. The resulting processed dataset is publicly available for research purposes [1] .

We now discuss two important observations from the dataset, that makes the task of attributing a observed network attack to a team difficult.

[1] https://cysis.engineering.asu.edu/cyber-attribution/

Table 2.2: Teams in the CTF competition

Notation	Team	Notation	Team
T-1	9447	T-11	clg
T-2	APT8	T-12	men in black hats
T-3	Alternatives	T-13	more smoked leet chicken
T-4	PPP	T-14	pwnies
T-5	Robot Mafia	T-15	pwningyeti
T-6	Samurai	T-16	routards
T-7	The European Nopsled Team	T-17	raon_ASRT (whois)
T-8	WOWHacker-BIOS	T-18	shell corp
T-9	[Technopandas]	T-19	shellphish
T-10	blue lotus	T-20	sutegoma2

Deception: In the context of this work we define an attack to be deceptive when multiple adversaries get mapped to a single attack pattern. In the current setting we define deception as the scenario when the same exploit is used by multiple teams to target the same team. Fig. 2.1 shows the distribution of unique deception attacks with respect to the total unique attacks in the dataset based on the target team. These unique deceptive attacks amount to just under 35% of the total unique attacks.

Duplicate attacks: A duplicate attack occurs when the same team uses the same payload to attack a team at different time instances. Duplicate attacks can be attributed to two reasons. First when a team is trying to compromise other system, it just does not launch a single attack but a wave of attacks with very little time difference between consecutive attacks. Second, once a successful payload is created which can penetrate the defense of other systems, it is used more by the original attacker as well as the deceptive one as compared to other payloads. We group duplicates as being non-deceptive and deceptive. Non-deceptive

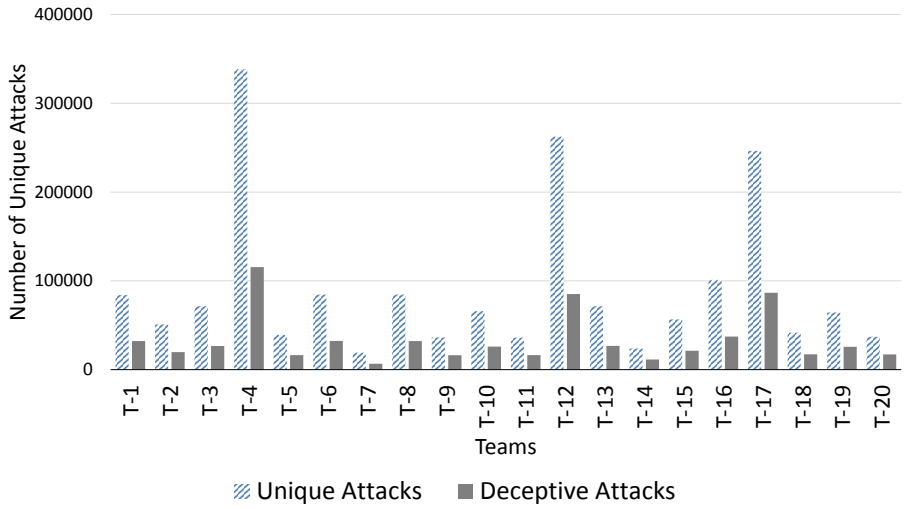


Fig. 2.1: Unique deceptive attacks directed towards each team

duplicate are the duplicates of the team that first initiated the use of a particular payload. On the other hand deceptive duplicates are all the attacks from the teams that are being deceptive. Deceptive duplicates form a large portion of the dataset as seen in Fig. 2.2.

Analyzing the number of teams that use a particular payload, gives us insights into the deceptive behavior of teams. We plot the usage of unique payloads with respect to the number of teams using them in their attacks. We use 4 different categories namely payloads used by a single team, payloads used by two teams, payloads used by three teams and payloads used by more than three teams. Fig. 2.3 shows the plot for each target team. A large fraction of unique payloads fall in the first two categories (one team and two teams).

2.3 Baseline Approaches

From the dataset we have the ground truth available for all the samples. Hence we use supervised machine learning approaches to predict the attacking team. The ground truth corresponds to a team mentioned in Table 2.2.

Decision Tree (DT). For baseline comparisons we first implemented a decision tree classifier.

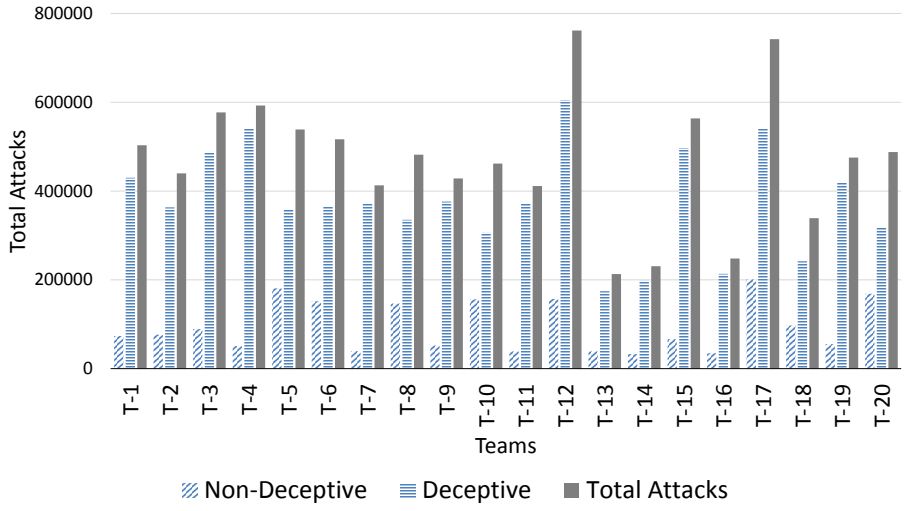


Fig. 2.2: Total attacks and duplicate attacks(Deceptive and Non-deceptive) directed towards each team

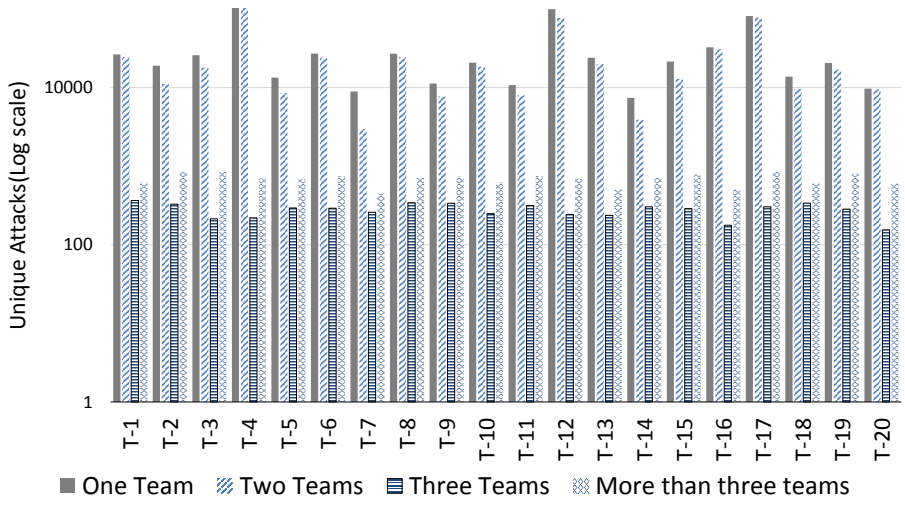


Fig. 2.3: Attacks on each target team by one team, two teams, three teams and more than three teams.

This hierarchical recursive partitioning algorithm is widely used for classification problems. We built the decision tree by finding the attribute that maximizes information gain at each split. This attribute is termed as the best split attribute and is used to split the node. Higher the information gain , the more pure the nodes that are split will be. During the testing phase, we check the test sample for the presence or absence of the best split attribute at each node till we reach the leaf node. The team that has majority samples at the leaf node, is predicted as the attack team for the test sample. In order to avoid over-fitting we terminate the tree, when the number of samples in the node are less than 0.1% of the training data.

Random Forest (RF). Random forest is an ensemble method proposed by Breiman [18]. It is based on the idea of generating multiple predictors which are then used in combination to classify unseen samples. The strength of random forest lies in injecting randomness to build each classifier and using random low dimensional subspaces to split the data at each node in a classifier. We use a random forest which combines bagging [18] for each tree with random feature selection [17] at each node to split the data thus generating multiple decision tree classifiers. To split the data at each node we use information gain with random subspace projection. The information gain indicates the amount of purity in the node with respect to class labels. More pure nodes result in higher information gain. Hence we try to find the splits that maximize the information gain. The advantage of using random forest over a single decision tree is low variance and the notion that weak learners when combined together have a strong predictive power. During the test phase , each test sample gets a prediction from each individual decision tree (weak learner) giving its own opinion on test sample. The final decision is made by a majority vote among those trees.

Support Vector Machine (SVM). Support vector machines is a popular supervised classification technique proposed by Vapnik [30]. SVM's works by finding a separating margin that maximizes the geometric distance between classes. The separating margin is termed as a hyperplane. We use the popular LibSVM implementation [25] which is publicly available.

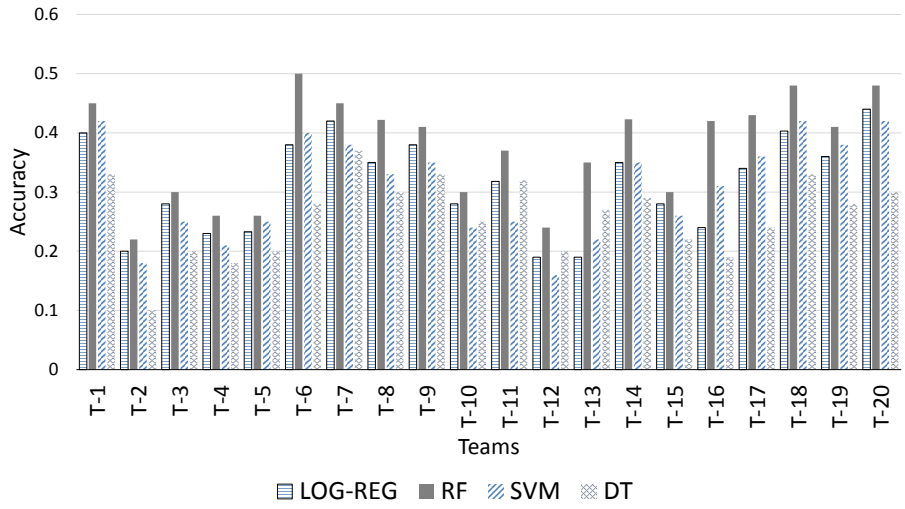


Fig. 2.4: Team prediction accuracy for LOG-REG, RF, SVM and DT.

SVM is inherently a binary classifier, and it deals with multi-class classification problems by implementing several 1-vs-1 or 1-vs-all binary classifiers which adds to the complexity as the number of classes increases.

Logistic Regression (LOG-REG). Logistic regression classifies samples by computing the odds ratio. The odds ratio gives the strength of association between the features and the class. As opposed to linear regression , the output of logistic regression is the class probability of the sample belonging to that class. We implement the multinomial logistic regression which handles multi-class classification.

2.3.1 Experimental Results

For our baseline experiments, we separate the attacks based on the team being targeted. Thus we have 20 attack datasets. We then sort the attack according to time. We reserve the first 90% of the attacks for training and the remaining 10% for testing. Attacker prediction accuracy is used as the performance measure for the experiment. Accuracy is defined as the fraction of correctly classified test samples. Fig. 2.4 shows the accuracy for predicting the

attacker for each target team. Machine learning techniques significantly outperform random guessing which would have an average accuracy of choosing 1 out of 19 teams attacking yielding an accuracy of 0.053. For this experiment random forest classifier performs better than logistic regression, support vector machine and decision tree for all the target teams. Table 2.3 below summarizes the average performance for each method.

Table 2.3: Summary of Prediction results averaged across all Teams for baseline models.

Method	**Average Performance**
Decision tree (DT)	0.26
Logistic regression (LOG-REG)	0.31
Support vector machine (SVM)	0.30
Random Forest (RF)	**0.37**

2.3.2 Misclassified Samples

Misclassification can be attributed to the following sources,

- Non-deceptive duplicate attacks attributed to one of the deceptive teams.
- Deceptive duplicates attributed to some other deceptive team.
- Payloads that were not encountered during the training phase.

The first two sources of error make up the majority of misclassifications, since a given attack can be attributed to any of the 19 teams.

Fig. 2.5 shows the distribution of the above mentioned sources of misclassification for each team. Deceptive duplicates form the majority of misclassifications. This is not surprising given the fact that deceptive duplicates make up almost 90% of the total attacks (see Fig. 2.2).

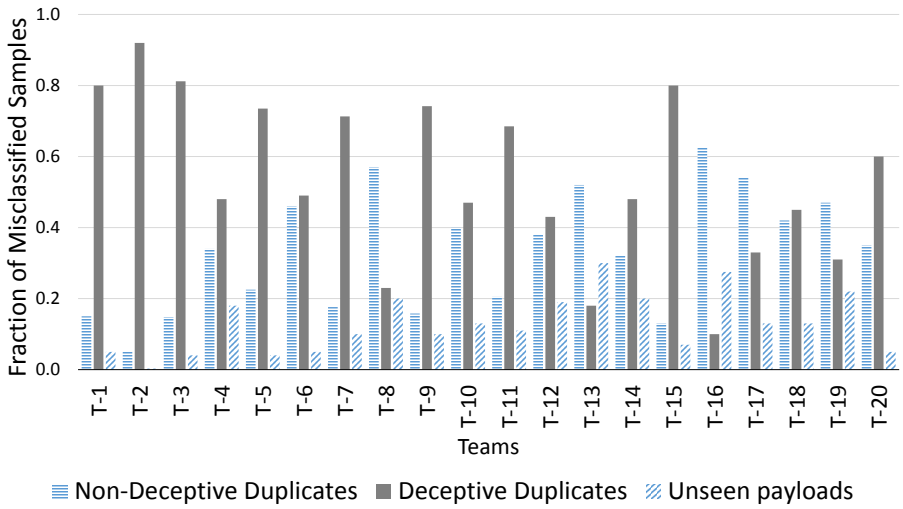


Fig. 2.5: Sources of error in the misclassified samples.

2.3.3 Average Prediction Probability

Fig. 2.6 shows the average probability of correctly classified and misclassified samples. The reported average probabilities are from the random forest classifier which performs the best among the baseline approaches (see Table 2.3). To compute this average probability we look at the predicted probability for each test sample rather than the prediction. For random forest the predicted probability is the average of individual decision trees in the forest. It is clear that the classifier predicts the correct team with higher probability as opposed to misclassified samples which are predicted with less confidence (probability).

2.4 Pruning

We explore different pruning techniques to address misclassification issues with respect to deceptive and non-deceptive duplicates. The pruning techniques are only applied to the training data, while the test data is maintained at 10% as mentioned in Section 2.3.1. We use the random forest classifier for all the pruning techniques.

These pruning techniques are briefly described as follows,

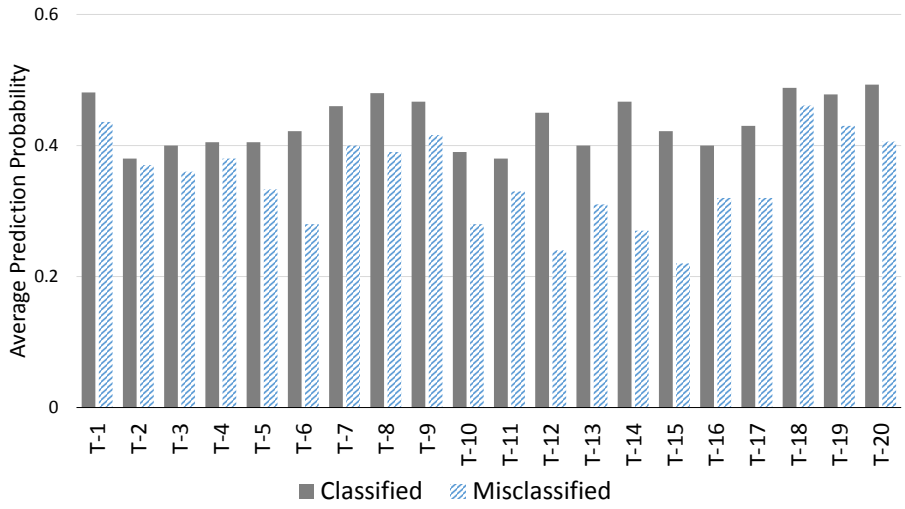


Fig. 2.6: Average Prediction probability for correctly classified and misclassified samples.

- *All-but-majority:* In this pruning we only consider the duplicates of the most attacking team given a payload and prune other duplicates.
- *All-but-K-majority:* Only consider the duplicates of the top K most frequent attacks given a payload and prunes the rest of the duplicates.
- *All-but-earliest-majority:* We only retain the duplicates of the team that initiates the attack given a payload, rest all duplicates are pruned.
- *All-but-most-recent-majority:* In this pruning we retain the duplicates of the team that last used the payload in the training data, rest all duplicates are pruned.

Table 2.4 gives the summary of the prediction results for all the pruning techniques in comparison with the random forest baseline approach. In the pruning techniques All-but-K-majority works best with an average accuracy of 0.42.

All-but-majority (P-1): In this pruning technique, for each payload, we only retain duplicates of the most frequent attacking team and prune the duplicates of all other teams. This pruned set is then used to train the random forest classifier. Table 2.5 shows the classifier

Table 2.4: Summary of Prediction results averaged across all Teams using pruning techniques.

Method	**Average Performance**
Baseline Approach (RF)	0.37
All-but-majority Pruning (RF)	0.40
All-but-K-majority Pruning (RF)	**0.42**
All-but-earliest Pruning (RF)	0.34
All-but-most-recent Pruning (RF)	0.36

performance in comparison with the baseline method. All-but-majority pruning technique has better performance on the test set than the baseline approach for 11 out of 20 teams. Using this pruning technique does benefit majority of the teams as the prediction accuracy improves for them, but for some teams the performance drops. The reason for the drop in performance for some teams is due to the fact that training set gets dominated by a single team which does not have majority in testing set. Since the majority team gets represented in most of the leaves of the random forest classifier, it gets predicted more often leading to high misclassifications.

All-but-K-majority (P-2): In order to address the issue of one team dominating in the training set, we use the all-but-K-majority where we consider the K most frequent teams for a payload under consideration. After trying out different values of K we select K = 3, which gives the best performance. For higher values of K, the pruning behaves like the baseline approach and for lower values it behaves like All-but-majority. On average each team gains about $40K$ samples in the training set as compared to all-but-majority pruning. Table 2.5 shows the classifier performance. In this case also pruning performs better than baseline in 11 out of 20 teams, but as compared to all-but-majority the performance for most teams is better.

All-but-earliest (P-3): For this pruning we only retain the duplicates of the team that initiated the attack using a particular payload. This pruning technique retains all the non-deceptive duplicates while getting rid of the deceptive ones. Table 2.5 shows the classifier performance. This pruning technique performs better than the baseline approach for 8 out of 20 teams. Comparing this result to all-but-majority (including all-but-K-majority) pruning indicates that deceptive duplicates are informative in attributing an attack to a team and should not be ignored completely.

All-but-most-recent (P-4): In this pruning we repeat a similar procedure like All-but-earliest but instead of retaining the duplicates of the team that initiated an attack, we retain the duplicates of the team that used the payload last in the training set. Because the data is sorted according to time, the last attacker becomes the most recent attacker for the test set. Table 2.5 shows the classifier performance.

2.4.1 Discussion

On further analysis of the misclassified samples from all-but-earliest and all-but-most-recent provides an interesting observation. Though majority of the misclassified samples between the two pruning techniques are similar, there is a fraction of samples which were correctly classified by all-but-earliest but misclassified by all-but-most-recent and vice versa. Let first (correct) denote the number of samples that were correctly classified from the misclassified samples of all-but-most-recent majority pruning experiment. Similarly last (correct) be the samples that were correctly classified from the misclassified samples of all-but-earliest majority pruning technique. Fig. 2.7 shows the number of samples that all-but-earliest pruning was able to classify correctly that were misclassified by all-but-most-recent. Fig. 2.8 shows a similar result for the other case. For both cases the correctly classified samples make up around 5-10% of the misclassified samples for each team. This result shows that using the two pruning techniques together to make attribution decision,

Table 2.5: Pruning technique performance comparison for each team.

Teams	**RF**	**P-1(RF)**	**P-2(RF)**	**P-3(RF)**	**P-4(RF)**
T-1	0.45	0.16	**0.46**	0.15	0.15
T-2	0.22	0.28	**0.30**	0.15	0.14
T-3	0.30	0.53	0.29	**0.57**	**0.57**
T-4	0.26	**0.33**	0.27	0.31	0.32
T-5	0.26	0.38	**0.45**	0.40	0.42
T-6	**0.50**	0.27	0.24	0.31	0.26
T-7	0.45	**0.59**	0.58	0.19	0.49
T-8	0.42	0.52	0.52	0.51	**0.55**
T-9	0.41	0.65	**0.68**	0.52	0.53
T-10	0.30	0.54	0.34	0.55	**0.57**
T-11	**0.37**	0.27	0.35	0.27	0.29
T-12	0.24	**0.37**	**0.37**	0.25	0.22
T-13	0.35	0.27	**0.37**	0.29	0.27
T-14	**0.42**	0.27	0.40	0.30	0.30
T-15	**0.30**	0.20	0.27	0.21	0.20
T-16	**0.42**	0.28	0.22	0.32	0.31
T-17	0.43	**0.45**	0.35	0.43	0.40
T-18	**0.48**	0.39	0.43	0.41	0.40
T-19	0.41	**0.65**	0.58	0.54	0.60
T-20	**0.48**	0.16	0.16	0.16	0.17

would lead to higher performance as opposed to using only one.

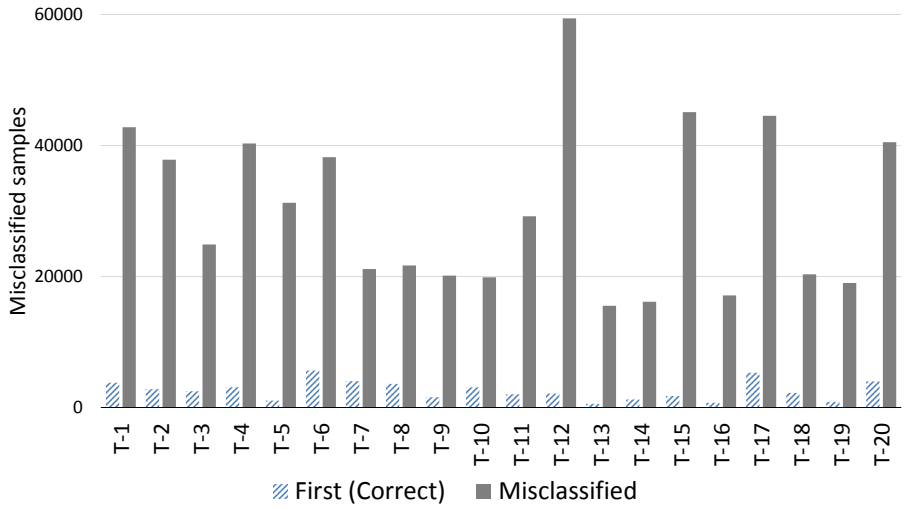


Fig. 2.7: Samples correctly classified by all-but-earliest and not by all-but-most-recent.

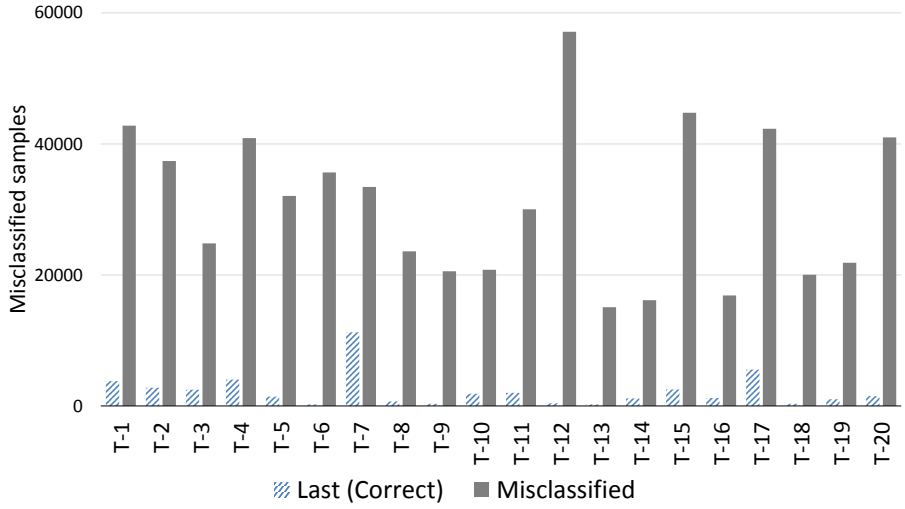


Fig. 2.8: Samples correctly classified by all-but-most-recent and not by all-but-earliest.

2.4.2 Ensemble Classifier

As discussed earlier using pruning techniques together would improve the prediction accuracy as opposed to using the pruning techniques individually. We perform an experiment to demonstrate it. We generate one prediction from each of the three pruning techniques namely, all-but-K-majority (since this pruning technique performs better than all-but-majority), all-but-earliest and all-but-most-recent. We call these predictions candidates for being the most likely attackers. We then define conditions to predict the actual attacker as follows:

1. *Predict the team that satisfies atleast two pruning cases.*

2. *If all the attacking candidates are different, then predict the all-but-K-majority prediction.*

Table 2.6 shows the results of this ensemble framework in comparison with the baseline approach and the different pruning techniques. The results are averaged across all teams. This 3-candidate ensemble framework performs the best with an average accuracy of **0.46** better than all the approaches indicating that using the pruning techniques in conjunction with each other improves the prediction accuracy.

2.5 Related Work

In our text on cyber-warfare [113], we discuss the difficulties of cyber-attribution and how an intelligence analyst must also explore the deception hypothesis in a cyber-warfare scenario. When compared to other domains of warfare, there is a much greater potential for evidence found in the aftermath of cyber-attack to be planted by the adversary for purposes of deception. The policy implications of cyber-attribution have also been discussed in [129] where the authors point out that anonymity, ability to launch multi-stage attacks, and attack

Table 2.6: Summary of Prediction results averaged across all Teams for ensemble classifier.

Method	**Average Performance**
Baseline Approach (RF)	0.37
All-but-majority Pruning (RF)	0.40
All-but-K-majority Pruning (RF)	0.42
All-but-earliest Pruning (RF)	0.34
All-but-most-recent Pruning (RF)	0.36
3-Candidate ensemble model (RF)	**0.46**

speed pose significant challenges to cyber attribution.

In an early survey on cyber-attribution [14], the authors point out that technical attribution will generally identify machines, as opposed to a given hacker and his/her affiliations. While we will use technical information in our approach, we have ground truth data on the group involved by the nature of the capture-the-flag data. This will allow our approach to profile the tactics, techniques, and procedures of a given group as we have ground-truth information on a hacking group as opposed to machines. An example of such an approach is the WOMBAT attribution method [34] which attributes behavior to IP sources that are potentially linked to some root cause determined through a clustering technique. Similarly, other work [126] combines cluster analysis with a component for multi-criteria decision analysis and studied an implementation of this approach using honeypot data again, this approach lacks any ground truth of the actual hacker or hacking group. In other work on attribution [58], the authors study the problem of attributing abnormal internal behavior to a malicious insider over the course of an advanced persistent threat (APT) a different type of attribution problem than the one we propose to study. Outside of cyber-security, attribution has also been studied in other contexts. Of particular note is the work of Walls [132]. Here, the author look at attribution based on forensic information in a much different problem. They consider

diverse sources, but do not seek to overcome inconsistency caused by intentional deception nor do they apply their methods to cyber-attacks. More recently the Q model has been proposed [102]. The framework of the Q model works by letting the analyst ask a range of relevant questions both technical and non-technical to aid in his process of attributing the attack. It provides a theoretical map towards cyber-attribution but does not address the issues of deception and does not evaluate the model on a relevant dataset.

Concurrently, we have devised a formal logical framework for reasoning about cyber-attribution [56, 114]. However, we have not studied how this framework can be instantiated on a real world dataset and, to date, we have not reported on an implementation or experiments in the literature.

We note that none of the previous work on cyber-attribution leverages a data set with ground truth information of actual hacker groups – which is the main novelty of this work.

2.6 Summary

In this chapter, we study cyber-attribution by examining DEFCON CTF data - which provides us with ground-truth on the culprit responsible for each attack. We frame cyber-attribution as a classification problem and examine it using several machine learning approaches. We find that deceptive incidents account for the vast majority of misclassified samples. Moving forward in Chapter 3, we look to employ a more principled approach to counter deception based on our previously established theoretical framework for reasoning about cyber-attribution [114]. In particular we wish to employ temporal reasoning to tackle the problem of deceptive attacks. This opens up interesting research questions in particular identifying hacking group from a series of attacks over a period of time, differentiating between deceptive hacking groups in time series data. This is a knowledge engineering challenge which calls for development of efficient and scalable reasoning framework.

Chapter 3

ARGUMENTATION MODELS FOR CYBER ATTRIBUTION

3.1 Introduction

A major challenge in cyber-threat analysis is to find the person or the group responsible for a cyber-attack. This is known as *cyber-attribution* [113, 22] and it is one of the central technical and policy challenges in cyber-security. Oftentimes, the evidence collected from multiple sources provides a contradictory viewpoint, which makes it unclear how the evidence needs to be combined or reasoned about in order to arrive at a conclusion. This gets worse in cases of *deception*, where either an attacker plants false evidence, or the evidence points to multiple threat actors, leading to uncertainty. In the text on cyber-warfare [113] the authors discuss the difficulties that an intelligence analyst faces in attributing an attack to a perpetrator given that deception might have occurred, and how the analyst needs to explore deception hypotheses under the given attack scenario. To resolve deception the analyst also needs to consider multiple sources of information—each with its level of confidence—to provide an adequate explanation for a particular decision made.

However, one of the main hurdles in the study and evaluation of cyber-attribution models is the lack of datasets with ground truth available regarding the party responsible for the attack—this has limited the evaluation of previous proposals. To overcome this, we built and leveraged a dataset from the capture-the-flag event held at DEFCON. In chapter 2, we used this dataset [1] to study cyber-attribution framed as a multi-class classification problem to predict the attacker. The machine learning model was able to achieve an accuracy of 37%, struggling in situations of deception, where similar attributes point towards multiple

[1] https://cysis.engineering.asu.edu/cyber-attribution/

attackers, as well as on previously unobserved attacks (attacks not encountered in the training data). We thus require a system that must be able to accomplish several goals, including:

- Reason about evidence in case of deceptive and previously unobserved attacks by relying on explanations (or arguments) constructed from information in the knowledge base.

- Integrate machine learning tools, towards a hybrid approach that allows to draw conclusions based on combinations of pieces of evidence and explanations constructed from the knowledge base.

- Provide analysts with indications as to how the system arrived at a particular conclusion (in this case the actor responsible for the attack), or why other conclusions were discarded.

To address these requirements, we have proposed a structured argumentation system that also integrates machine learning approaches in order to reason about cyber events of interest and tackle the cyber attribution problem. This is—to the best of our knowledge—the first line of research that combines a Knowledge Representation (KR) formalism, in the form of structured argumentation, with machine learning (ML) to address cyber attribution. This chapter brings together and extends the results of chapter 2, combining the DeLP formalism with machine learning to significantly improve accuracy. This chapter includes:

- A hybrid KR-ML framework that combines machine learning techniques with defeasible argumentation for cyber attribution.

- The construction and analysis of a set of argumentation models built on top of each other in order to improve the performance of the machine learning model.

- An empirical evaluation of the system on our curated dataset; our experiments show

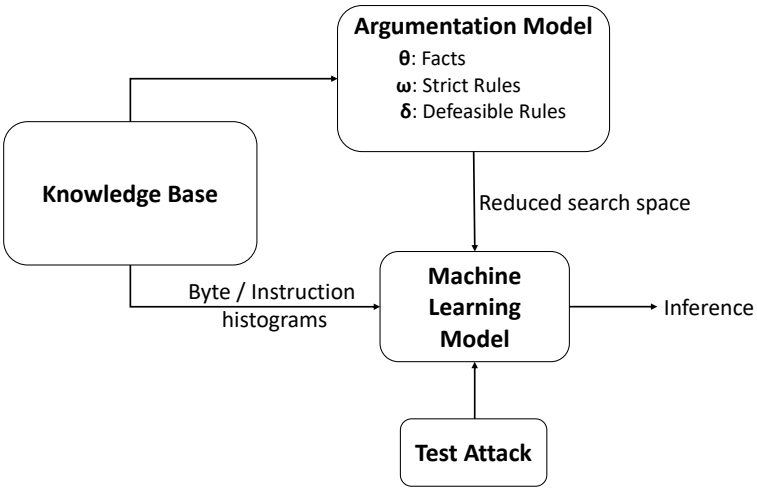


Fig. 3.1: Reasoning system

that the hybrid approach achieves a significant accuracy of 64.5%, as compared to 37% achieved by machine learning approaches alone.

- A detailed discussion about the relevance of rules introduced in each of the proposed argumentation models. The discussion is supported by performance numbers demonstrating the rules that have the most significant impact on the system's performance.

3.2 System Overview

Fig. 3.1 gives an overview of the reasoning system we developed; it consists of the following three main modules:

- **Knowledge Base:** Stores evidence of previously conducted attacks with the ground truth of the person/group responsible for the attack—we use this content as the training data. The data can be of different types, including the tools/softwares used in the attack, attacker behavior information (including motives for the attack), and so on. In this chapter, the knowledge base is comprised of all the network traffic captured

during the capture-the-flag competition. In these competitions there are 20 teams (see Table 2.2) responsible for conducting cyber attacks by exploiting vulnerabilities in each other's system. These 20 teams are responsible for the 10 million attacks that make up the dataset. Each attack is represented by the attacking team, the target team, instruction and byte histograms of the exploit used in the attack, as well as the timestamp of the attack (see Table 2.1). We discuss the dataset and the DEFCON CTF event in detail in Section 2.2. We sort the dataset by time and divide it according to target teams listed in Table 2.2. The first 90% is reserved for training (knowledge base) and the remaining 10% for testing.

- **Argumentation Model:** This component constructs arguments for a given query (attack of interest) using elements in the knowledge base. We use a formalism that combines logic programming with defeasible argumentation (DeLP). It is made up of three constructs: *facts:* observations from the knowledge base that cannot be contradicted; *strict rules:* logical combinations of facts that are always true; and *defeasible rules:* can be thought of as strict rules but are only true if no contradictory evidence is present. We discuss the argumentation framework with examples for each of the constructs in Section 3.3. Arguments help reduce the set of possible attackers behind the attack; this reduced set of possible attackers acts as one of the inputs to the machine learning model. The argumentation model thus constrains the machine learning model to predict the attacker from the reduced set of possible attackers.

- **Machine learning model:** The machine learning model takes the knowledge base and query as input, along with the reduced set of possible attackers from the argumentation model, and provides an attacker deemed responsible for the attack. It is constrained by the argumentation model to select the attacker from the reduced attacker set, which aids the machine learning model as demonstrated in the results section of the chapter.

We use the byte and instruction histograms extracted from the exploits used in the attack as features for the machine learning model. Any standard machine learning model can be used in this module, based on the type of attack data being analyzed. Here, we use random forest, which have exhibited the best performance for this dataset [91].

3.3 Argumentation Model

Our approach relies on a model of the world where we can analyze competing hypotheses in a cyber-operation scenario. It should allow for contradictory information so it can handle inconsistency in cases of deception. In his section, we review the necessary ideas from defeasible logic-based argumentation [42], and provide examples of this approach instantiated for our cyber attribution problem.

Before describing the argumentation model in detail, we introduce some necessary notation. Variables and constant symbols represent items such as the exploits/payloads used for the attack, and the actors conducting the cyber-attack (in this case, the teams in the CTF competition). We denote the set of all variable symbols with $\mathbf{V}$ and the set of all constants with $\mathbf{C}$. In the running example, we use a subset of our DEFCON CTF dataset. For our model we require two subsets of $\mathbf{C}$: $\mathbf{C}_{act}$, denoting the actors capable of conducting the cyber-operation, and $\mathbf{C}_{exp}$, denoting the set of unique exploits used. We use strings starting with capital letters to denote variables.

Example 1. *The following are examples of actors and cyber-operations from the CTF data:* $\mathbf{C}_{act} = \{\textit{bluelotus, robotmafia, apt8}\}$, $\mathbf{C}_{exp} = \{\textit{exploit}_1, \textit{exploit}_2, ..., \textit{exploit}_n\}$.

The language also contains a set of predicate symbols that have constants or variables as arguments, and denote events that can be either *true* or *false*. A ground atom is composed by a predicate symbol and a tuple of constants, one for each argument. The set of all ground

atoms is denoted with $\mathbf{G}$. A ground literal L is a ground atom or a negated ground atom. An example of a ground atom for our running example is $\mathit{attack}(\mathit{exploit}_1, \mathit{bluelotus})$. We denote a subset of $\mathbf{G}$ with $\mathbf{G}'$. We denote the set of predicates with $\mathbf{P}$.

Example 2. *Some examples of predicates are shown in Table 3.1.*
For instance, culprit$(\mathit{exploit}_1, \mathit{apt8})$ *will either be true or false, and denotes the event where apt8 used* $\mathit{exploit}_1$ *to conduct a cyber-operation.*

We define deceptive and replay attacks in the context of this work:

Deception: In the context of this work, we define an attack to be *deceptive* when multiple adversaries get mapped to an identical exploit. In the current setting we define deception as the scenario in which the same exploit is used by multiple teams against the same target. In Table 3.1, $\mathsf{deception}(\mathit{exploit}_1, apt8)$ denotes the event where *apt8* used $\mathit{exploit}_1$ to conduct a deceptive attack.

Replay attacks: A *replay* attack occurs when the a team uses the same payload at different points in time. In Table 3.1, $\mathsf{replay\_attack}(\mathcal{E}, Y)$ denotes the event where exploit $\mathcal{E}$ was replayed by team *Y*. Replay attacks can be attributed to two reasons. First, when a team is trying to compromise another's system, it does not just launch a single attack but rather a wave of attacks with very little time difference in between consecutive attacks. Second, once a successful payload is created that can penetrate the defense of other systems, it is used both by the original attacker as well as the deceptive ones, and it is used in more occasions compared to other payloads. We group replay attacks as either being *non-deceptive* or *deceptive*. The former are copies of the attacks launched by the team that first initiated the use of a particular payload; on the other hand, deceptive replay attacks are all the attacks from the teams that did not initiate the use of a given payload.

We choose a structured argumentation framework [101] for our model; our approach works by creating arguments (in the form of a set of rules and facts) that compete with each

Table 3.1: Example Predicates and explanation

Predicate	**Explanation**
attack($exploit_1$, *bluelotus*)	$exploit_1$ was targeted towards the team Blue Lotus.
replay_attack($\mathcal{E}$, *Y*)	Exploit $\mathcal{E}$ was replayed by team *Y*.
deception($exploit_1$, $apt8$)	Team *apt8* used $exploit_1$ for deception.
time_diff(I, *Y*)	Team *Y* was deceptive within the given time interval I.
culprit($exploit_1$, *apt8*)	Team *apt8* is the likely culprit for the attack (using $exploit_1$ on the target team).

other to attribute an attack to a given perpetuator. In this case, arguments are defeated based on contradicting information in other arguments. This procedure is known as a *dialectical process*, where the arguments that are undefeated prevail—such arguments are said to be *warranted*, and they give a clear map of what conclusions are adequately supported. This transparency lets a security analyst not only add new arguments based on new evidence discovered in the system, but also get rid of incorrect information and fine-tune the model for better performance. Since the argumentation model can deal with inconsistent information, it draws a natural analogy to the way humans settle disputes when there is contradictory information available. Having a clear explanation of why one argument is chosen over others is a desirable characteristic for both analysts and organizations that need to make decisions and policy changes.

3.3.1 Defeasible Logic Programming (DeLP)

DeLP is a formalism that combines logic programming with defeasible argumentation; we will now provide a brief introduction, but full details can be found in [42]. The formalism is made up of several constructs, namely facts, strict rules, and defeasible rules. Facts represent statements obtained from evidence, and are always true; similarly, strict rules

are logical combinations of facts that always hold. On the contrary, defeasible rules can be thought of as strict rules that may be true in some situations, but could be false if contradictory evidence is present. These three constructs are used to build arguments, and DeLP programs are sets of facts, strict rules and defeasible rules. We use the usual notation for DeLP programs, denoting the knowledge base with $\Pi = (\Theta, \Omega, \Delta)$, where Θ is the set of facts, Ω is the set of strict rules, and Δ is the set of defeasible rules. Examples of the three constructs are provided with respect to the dataset in Fig. 3.2. We now describe the constructs in detail.

Facts (Θ) are ground literals that represent atomic information or its (strong) negation ($\neg$).

Strict Rules (Ω) represent cause and effect information; they are of the form $L_0 \leftarrow L_1, ...L_n$, where L_0 is a literal and $\{L_i\}_{i>0}$ is a set of literals.

Defeasible Rules (Δ) are weaker versions of strict rules, and are of the form $L_0 \prec L_1, ..., L_n$, where L_0, is the literal and $\{L_i\}_{i>0}$ is a set of literals.

When a cyber-attack occurs, the model can be used to derive arguments as to who could have conducted the attack. Derivation follows the same mechanism as logic programming [73]. DeLP incorporates defeasible argumentation, which decides which arguments are warranted and it blocks arguments that are in conflict and a winner cannot be determined.

Example 3. *Fig. 3.2 shows a ground argumentation framework demonstrating constructs derived from the CTF data. For instance, θ_1 indicates the fact that exploit$_1$ was used to target the team Blue Lotus, and θ_5 indicates that team pwnies is the most frequent user of exploit$_1$. For the strict rules, ω_1 says that for a given exploit$_1$ the attacker is pwnies if it was the most frequent attacker and the attack exploit$_1$ was replayed. Defeasible rules can be read similarly; δ_2 indicates that exploit$_1$ was used in a deceptive attack by APT8 if it was replayed and the first attacker was not APT8. By replacing the constants with variables in the predicates we can derive a non-ground argumentation framework.*

Θ : $\theta_1 =$ attack($exploit_1, bluelotus$)

$\theta_2 =$ first_attack($exploit_1, robotmafia$)

$\theta_3 =$ last_attack($exploit_1, apt8$))

$\theta_4 =$ time_diff($interval, robotmafia$)

$\theta_5 =$ most_frequent($exploit_1, pwnies$)

Ω : $\omega_1 =$ culprit($exploit_1, pwnies$) $\leftarrow$ most_frequent($exploit_1, pwnies$), replay_attack($exploit_1$)

$\omega_2 =$ $\neg$ culprit($exploit_1, robotMafia$) $\leftarrow$ last_attack($exploit_1, apt8$), replay_attack($exploit_1$)

Δ : $\delta_1 =$ replay_attack($exploit_1$) $\prec$ attack($exploit_1, bluelotus$), last_attack($exploit_1, apt8$)

$\delta_2 =$ deception($exploit_1, apt8$) $\prec$ replay_attack($exploit_1$), first_attack($exploit_1, robotmafia$)

$\delta_3 =$ culprit($exploit_1, apt8$) $\prec$ deception($exploit_1, apt8$), replay_attack($exploit_1$)

$\delta_4 =$ $\neg$culprit($exploit_1, apt8$) $\prec$ time_diff($interval, robotmafia$)

Fig. 3.2: A ground argumentation framework.

Definition 1. ***(Argument)*** *An argument for a literal L is a pair $\langle \mathcal{A}, L \rangle$, where $\mathcal{A} \subseteq \Pi$ provides a minimal proof for L meeting the requirements: (1) L is defeasibly derived from $\mathcal{A}$ [2] , (2) $\Theta \cup \Omega \cup \mathcal{A}$ is not contradictory, and (3) $\mathcal{A}$ is a minimal subset of Δ satisfying 1 and 2, denoted $\langle \mathcal{A}, L \rangle$.*

Literal L is called the *conclusion* supported by the argument, and $\mathcal{A}$ is the *support*. An

[2]Defeasible derivations are sequences of rules (possibly including defeasible rules) that end in L.

$\langle \mathcal{A}_1, \mathsf{replay\_attack}(\mathit{exploit}_1) \rangle$	$\mathcal{A}_1 = \{\delta_1, \theta_1, \theta_3\}$
$\langle \mathcal{A}_2, \mathsf{deception}(\mathit{exploit}_1, \mathit{apt8}) \rangle$	$\mathcal{A}_2 = \{\delta_1, \delta_2, \theta_2\}$
$\langle \mathcal{A}_3, \mathsf{culprit}(\mathit{exploit}_1, \mathit{apt8}) \rangle$	$\mathcal{A}_3 = \{\delta_1, \delta_2, \delta_3\}$
$\langle \mathcal{A}_4, \neg\mathsf{culprit}(\mathit{exploit}_1, \mathit{apt8}) \rangle$	$\mathcal{A}_4 = \{\delta_1, \delta_4, \theta_3\}$

Fig. 3.3: Example ground arguments from Fig. 3.2.

argument $\langle \mathcal{B}, L \rangle$ is a *subargument* of $\langle \mathcal{A}, L' \rangle$ iff $\mathcal{B} \subseteq \mathcal{A}$. The following examples show arguments for our scenario.

Example 4. *Fig. 3.3 shows example arguments based on the knowledge base from Fig. 3.2; here,* $\big\langle \mathcal{A}_1$, $\mathsf{replay\_attack}(\mathit{exploit}_1) \big\rangle$ *is a subargument of* $\big\langle \mathcal{A}_2$, $\mathsf{deception}(\mathit{exploit}_1, \mathit{apt8}) \big\rangle$ *and* $\big\langle \mathcal{A}_3$, $\mathsf{culprit}(\mathit{exploit}_1, \mathit{apt8}) \big\rangle$.

For a given argument there may be counter-arguments that contradict it. For instance, referring to Fig. 3.3, we can see that $\mathcal{A}_4$ attacks $\mathcal{A}_3$. A *proper defeater* of an argument $\langle A, L \rangle$ is a counter-argument that—by some criterion—is considered to be better than $\langle \mathcal{A}, L \rangle$; if the two are incomparable according to this criterion, the counterargument is said to be a *blocking* defeater. The default criterion used in DeLP for argument comparison is *generalized specificity* [119].

A sequence of arguments is called an *argumentation line*. There can be more than one defeater argument, which leads to a tree structure that is built from the set of all argumentation lines rooted in the initial argument. In this *dialectical tree*, every child can defeat its parent (except for the root), and the leaves represent the undefeated arguments; this creates a map of all possible argumentation lines that decide if an argument is defeated or not. Arguments that either have no attackers or all attackers have been defeated are said to be *warranted*.

Given a literal L and an argument $\big\langle \mathcal{A}, L \big\rangle$, in order to decide whether or not a literal

L is warranted, every node in the dialectical tree $\mathcal{T}(\langle \mathcal{A}, L \rangle)$ is recursively marked as "D" (*defeated*) or "U" (*undefeated*), obtaining a marked *dialectical tree* $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$ where:

- All leaves in $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$ are marked "U", and
- Let $\langle \mathcal{B}, q \rangle$ be an inner node of $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$. Then, $\langle \mathcal{B}, q \rangle$ will be marked "U" iff every child of $\langle \mathcal{B}, q \rangle$ is marked "D". Node $\langle \mathcal{B}, q \rangle$ will be marked "D" iff it has at least one child marked "U".

Given argument $\langle \mathcal{A}, L \rangle$ over Π, if the root of $\mathcal{T}^*(\langle \mathcal{A}, L \rangle)$ is marked "U", then $\mathcal{T}^*(\langle \mathcal{A}, h \rangle)$ *warrants* L and that L is *warranted* from Π. (Warranted arguments correspond to those in the grounded extension of a Dung argumentation system [37].)

In practice, an implementation of DeLP accepts as input sets of facts, strict rules, and defeasible rules. Note that while the set of facts and strict rules is consistent (non-contradictory), the set of defeasible rules can be inconsistent. We engineer our cyber-attribution framework as a set of defeasible and strict rules whose structure was created manually, but are dependent on values learned from a historical corpus. Then, for a given incident, we instantiate a set of facts for that situation. This information is then provided as input into a DeLP implementation that uses heuristics to generate all arguments for and against every possible culprit for the cyber attack. Dialectical trees based on these arguments are analyzed, and a decision is made regarding which culprits are *warranted*. This results in a reduced set of potential culprits, which we then use as input into a classifier to obtain an attribution decision.

3.4 Models

We now introduce several DeLP-based models that capture important aspects relating to cyber attribution and can be instantiated using available data.

$$\Theta: \quad \theta_1 = \text{attack}(\mathcal{E}, X)$$
$$\theta_2 = \text{first\_attack}(\mathcal{E}, Y)$$
$$\theta_3 = \text{last\_attack}(\mathcal{E}, Y)$$

Fig. 3.4: Facts defined for each test sample.

3.4.1 Baseline Argumentation Model (BM)

We use the following notation: let $\mathcal{E}$ be the attack query under consideration aimed at target team *X*, *Y* represent all the possible attacking teams, and $\mathcal{D}$ be the set of all deceptive teams (those using deceptive attacks) if the given attack is deceptive in the training set. For non-deceptive attacks, $\mathcal{D}$ will be empty. We note that facts cannot have variables, only constants (however, help compress the program for readability purposes, we use *meta-variables* in facts). To begin, we define the facts described in Fig. 3.4; fact θ_1 states that attack $\mathcal{E}$ was used to target team *X*, θ_2 states that team *Y* was the first team to use the attack $\mathcal{E}$ in the training data, and similarly θ_3 states that team *Y* was the last team to use the attack $\mathcal{E}$ in the training data. The first and last attacking team may or may not be the same. We study the following three cases:

Case 1: Non-deceptive attacks. In non-deceptive attacks, only one team uses the payload to target other teams in the training data. It is easy to predict the attacker for these cases, since the search space only has one team. To model this situation, we define a set of defeasible and strict rules: In Fig. 3.5, defeasible rule δ_1 checks whether the attack was replayed in the training data. Since it is a non-deceptive attack, it is only replayed by the same team. The strict rule ω_1 then puts forth an argument for the attacker (culprit) if the defeasible rule holds and there is no contradiction for it.

Case 2: Deceptive attacks. These attacks form the majority of the misclassified samples in previous work [91]. In this case, set $\mathcal{D}$ is not empty; let $\mathcal{D}_i$ denote the deceptive teams

Ω : $\omega_1 =$ culprit($\mathcal{E}$, Y) $\leftarrow$ last_attack($\mathcal{E}$, Y), replay_attack($\mathcal{E}$).

Δ : $\delta_1 =$ replay_attack($\mathcal{E}$) $\prec$ attack($\mathcal{E}$, X), last_attack($\mathcal{E}$, Y).

Fig. 3.5: Defeasible and strict rule for non-deceptive attack.

Θ : $\theta_4 =$ decep($\mathcal{E}$, X)

$\theta_5 =$ frequent ($\mathcal{E}$, F)

Ω : $\omega_2 =$ $\neg$culprit($\mathcal{E}$, Y) $\leftarrow$ first_attack($\mathcal{E}$, Y), decep($\mathcal{E}$, X)

Δ : $\delta_2 =$ replay_attack($\mathcal{E}$) $\prec$ attack($\mathcal{E}$, X), last_attack($\mathcal{E}$, Y)

$\delta_3 =$ deception($\mathcal{E}$, $\mathcal{D}_i$) $\prec$ replay_attack($\mathcal{E}$), first_attack($\mathcal{E}$, Y)

$\delta_4 =$ culprit($\mathcal{E}$, $\mathcal{D}_i$) $\prec$ deception($\mathcal{E}$, $\mathcal{D}_i$), first_attack($\mathcal{E}$, Y)

Fig. 3.6: Facts and rules for deceptive attacks.

in $\mathcal{D}$. We also compute the most frequent attacker from the training data given a deceptive attack, and denote it with F. The DeLP components that model this case include the ones in Fig. 3.6; fact θ_4 indicates that attack $\mathcal{E}$ was deceptive towards team *X*, and θ_5 indicates the most frequent attacker team F from the training data. Strict rule ω_2 indicates that in case of deception the first attack team *Y* is not the attacker. For the defeasible rules, δ_2 deals with the case in which attack $\mathcal{E}$ was replayed, δ_3 deals with the case of deceptive teams from the set $\mathcal{D}$, and δ_4 indicates that all the deceptive teams are likely to be the attackers in the absence of any contradictory information.

Case 3: Previously Unseen Attacks. The most difficult attacks to attribute in the dataset are the new ones, which are attacks first encountered in the test set (i.e., those that did not occur in the training set). To build constructs for this kind of attack we first compute the k

nearest neighbors from the training set according to a simple Euclidean distance between the byte and instruction histograms of the two attacks. For each of the matching attacks from the training data we check if the attack is deceptive or non-deceptive. If non-deceptive, we follow the procedure for Case 1, otherwise we follow the procedure for Case 2. Since we replace one unseen attack with three seen attacks, the search space for the attacker increases in these cases.

Attacker Time Analysis

The CTF data provides us with time stamps for the attacks in the competition. We can use this information to come up with rules for/against an argument for a team being the attacker. We compute the average time for a team to replay its own attack given that it was the first one to deploy it (see Fig. 3.7). It can be observed that teams like *more smoked leet chicken* (T-13) and *Wowhacker-bios* (T-8) are very quick to replay their own attacks as compared to other teams. Fig. 3.7 also shows the average time for a team to perform a deceptive attack. Teams like *The European* (T-7) and *Blue lotus* (T-10) are quick to commit deception, while others take more time.

We use this time information to narrow down our search space for possible attackers. In particular, for a deceptive attack query, we compute the time difference between the query and the training sample that last used the same payload. We denote this time difference as $\triangle t$, and include it as a fact θ_6. We then divide the deceptive times from Fig. 3.7 into appropriate intervals; each team is assigned to one of those time intervals. We then check which time interval $\triangle t$ belongs to and define a defeasible rule δ_5 that makes a case that all teams not belonging to the interval are not the culprits, as shown in Fig. 3.8.

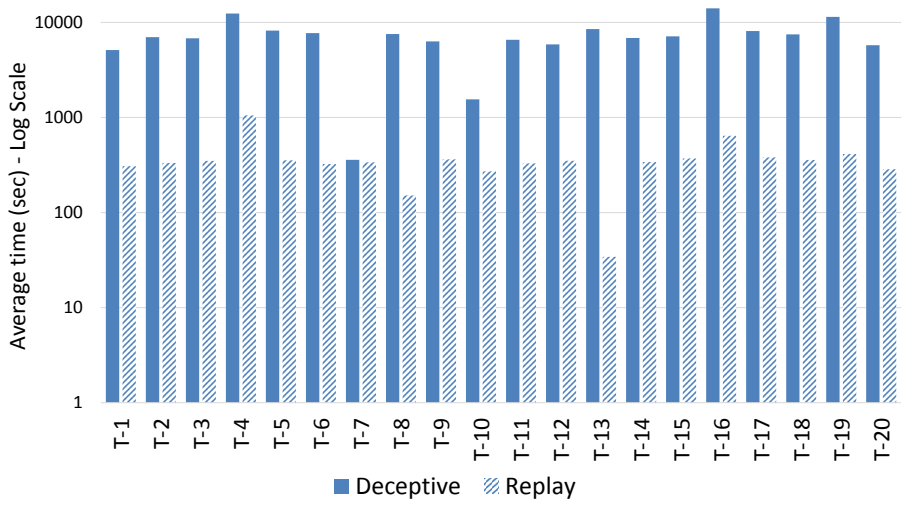


Fig. 3.7: Average time for teams to perform a deceptive attack and replay own attacks (Log-scale).

Θ : $\theta_6 =$ timedifference $(\mathcal{E}, X)$

For $Y \notin$ interval:

Δ : $\delta_5 =$ $\neg$culprit$(\mathcal{E}, Y) \prec$ timedifference $(\mathcal{E}, X)$.

Fig. 3.8: Time facts and rules. Interval indicates a small portion of the entire deceptive time (for instance, less than $2,000$ seconds, more than $8,000$ seconds, and so on).

3.4.2 Extended Baseline Argumentation Model I (EB1)

Previously unseen attacks make up almost 20% of the test samples for each target team. On analyzing the misclassification from the baseline argumentation model, we observe that the majority of these attacks get misclassified (>80%).

The misclassifications can be attributed to two main reasons: first, the reduced search space is not able to capture the ground truth for unseen attacks (i.e., the actual attacker is not present in the reduced search space), which leads to a wrong decision by the learning

model; second, since we represent each unseen attack by the k most similar attacks from the training data, this leads to an increase in the size of the search space—the presence of more choices for the learning model makes it more difficult to make the correct choice [94].

We address these issues by leveraging two sets of defeasible rules. First, for each target team we compute from the training set the top three teams that come up with the *most unique exploits*, as these teams are more likely to launch an unseen attack in the test set. The intuition behind this rule is that not all teams write their own exploits—most teams just capture a successful exploit launched by other teams and repackage it and use it as their own (deception). The second set of rules is proposed to avoid addition of *less similar teams* to the reduced search space. In the baseline model we use 3-nearest neighbors to represent an unseen attack; in this extended version, we consider only the nearest neighbors that are less than a particular threshold value T (set for each target team separately). So, each attack will be represented by $k \leq 3$ teams depending upon the threshold requirement. Apart from the baseline model rules, we propose additional rules for deceptive attacks. Let $\mathcal{U}$ denote the set of teams with the three highest numbers of unique attacks in the training data. Also, let $\mathcal{N}$ denote the set of three most similar culprits for the given unseen attack. The extended model is shown in Fig. 3.9; fact θ_7 indicates the teams present in $\mathcal{N}$ and whose similarity is less than a particular threshold T, while θ_8 indicates if the team u_i was one of most unique attackers from set $\mathcal{U}$. For the defeasible rules, δ_6 follows fact θ_7, stating that the teams in $\mathcal{N}$ that satisfy the threshold condition are likely to be the culprits, and δ_7 indicates that if u_i is a unique attacker then it can be the culprit unless contradictory information is available. $\mathcal{U}$ is independent of the test samples and will be the same for all previously unseen attacks given a target team.

For each of the similar payloads (three or fewer) computed from the training data we check if the attack is deceptive or non-deceptive. If non-deceptive, we follow the procedure for Case 1, otherwise we follow the procedure for Case 2 stated in the baseline argumentation

model.

$$
\begin{array}{lll}
 & \text{For } (n_i \in \mathcal{N} \text{ and } sim < T): & \\
\Theta: & \theta_7 = & \mathsf{threshold}(\mathcal{E}, T) \\
 & \text{For } u_i \; in \; \mathcal{U}: & \\
 & \theta_8 = & \mathsf{unique}(\mathcal{E}, u_i) \\
\hline
\Delta: & \delta_6 = & \mathsf{culprit}(\mathcal{E}, n_i) \prec \mathsf{threshold}(\mathcal{E}, T) \\
 & \text{For } u_i \in \mathcal{U}: & \\
 & \delta_7 = & \mathsf{culprit}(\mathcal{E}, u_i) \prec \mathsf{unique}(\mathcal{E}, u_i)
\end{array}
$$

Fig. 3.9: Rules for unseen attacks.

3.4.3 Extended Baseline Argumentation Model II (EB2)

Another source of misclassification in the baseline argumentation model is the presence of previously unseen deceptive teams and their replayed attacks. These refer to teams that did not use the exploit in the training set but started using it in the test set. It is difficult for a machine learning approach to pinpoint such a team as being the culprit if it has not encountered it using the exploit in the past. In our dataset, these attacks comprise 15% of the total, and up to 20% for some target teams.

In order to address this issue, we group together teams that have similar deceptive behavior based on the time information available to us from the training set; for instance, teams that are deceptive within a certain interval of time (e.g., less than 2,000 seconds) after the first attack has been played are grouped together. For a given attack query, we compute the time difference between the query and the last time the attack was used in the training set. We then assign this time difference to a specific group based on which interval the time difference falls in. In order to fine-tune the time intervals, instead of using the average

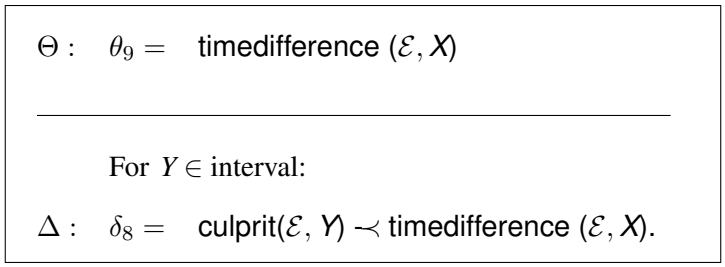


Fig. 3.10: Time facts and rules. Interval indicates a small portion of the entire deceptive time (for instance, less than $2,000$ seconds, more than $8,000$ seconds, and so on).

deceptive times averaged across all target teams (as used in the baseline model), we compute and use deceptive times for each target team separately. We model the time rules as stated in Fig. 3.10; fact θ_9 states the time difference between the query and the last training sample to use that attack. The defeasible rule δ_8 on the other hand states that teams belonging to that interval (in which the time difference lies) are likely to be the culprits unless a contradiction is present. It is clear that this rule will increase the search space for the query, as additional teams are now being added as likely culprits (see Table 3.11).

3.4.4 Extended Baseline Argumentation Model III (EB3)

We now describe a third approach to address misclassifications. In EB1, we try to address the issue by introducing rules that include additional teams in the reduced search space given their ability to create unique exploits and not mimic other teams. In EB2, similarly behaving teams are added to the search space based on the time rules to tackle unseen attacks. The rules constructed in the previous proposed models are with respect to a particular query, and do not incorporate the behavior of the attacking team towards different target teams irrespective of the query.

Analyzing the attack patterns for each team in the CTF event (using the knowledge base), preferred target teams for each team can be identified. Preferred teams have large

amounts of attacks directed towards them from a single team; this is not surprising given that technical expertise differs from team to team. Teams might find themselves target of a large number of attacks if they are not able to find the system vulnerabilities quickly and patch them up. Since teams have access to the network traffic at all times, they can identify vulnerable teams based on the network traffic directed towards any team. We leverage this information by identifying the teams that are responsible for a majority of the attacks given a target team. From this set we select teams based on a threshold set to more than 20% of the total attacks. We used different threshold values and selected 20%, as reducing the threshold adds more teams without improving the performance, resulting in a larger reduced search space and a degradation of the performance for some target teams. We denote this set as F, and the threshold as λ.

In the above-discussed time based rules, teams are blocked as being the likely attackers if the attack time difference for the teams falls outside the defined interval (see Fig. 3.8). This leads to eliminations of some of the deceptive teams that could have been the likely culprit. We construct defeasible rules by combining the time based rules with teams identified as preferred attackers for a given target team. The rules are similar to Fig. 3.8, with an additional condition added to not block the team if it is one of the preferred attackers for the target team.

Fig. 3.11 shows the updated time rules. We first compute the time difference between the query and the training sample that last used the same payload (denoted as $\triangle t$), and include it as a fact θ_{10}. Preferred attackers for the given target team *X* are then computed that satisfy the given threshold condition; this is represented as fact θ_{11}. We divide the deceptive times from Fig. 3.7 into appropriate intervals; each team is assigned to one of those time intervals. Then, we check which time interval $\triangle t$ belongs to, and define a defeasible rule δ_9 that makes a case for all teams not belonging to the interval to not be the culprits given that the team is not from the preferred attacker set F.

Θ : $\theta_{10} =$ timedifference ($\mathcal{E}$, X)

Θ : $\theta_{11} =$ pref_attacker(X, λ)

For $Y \notin$ interval and $Y \notin$ in F :

Δ : $\delta_9 =$ $\neg$culprit($\mathcal{E}$, Y) $\prec$ timedifference ($\mathcal{E}$, X).

Fig. 3.11: Time facts and rules. Interval indicates a small portion of the entire deceptive time (for instance, at most $2,000$ seconds, at least $8,000$ seconds, and so on), and F is the preferred attacker set for a given target team X.

3.5 Experimental Evaluation

We now report on the results of a series of experiments that we run in order to evaluate our proposed methodology. The dataset was discussed in chapter 2 (see Section 2.2).

3.5.1 Results

Table 3.2 shows the comparison of the four proposed models (baseline plus the three extended argumentation models) with respect to the reduced search space and the fraction that of the reduced search space that contain the ground truth. We observe that for EB3 the search space is increased by an average of almost 3.7 teams per test sample from EB1; at the same time the presence of ground truth in the reduced search space increased to 0.83, which is a significant improvement over 0.68.

Baseline Argumentation Model (BM)

We now provide a summary of the experimental results—the setup is similar to chapter 2: the dataset is sorted by time for each target team, the first 90% of the data is used for training and the remaining 10% for testing. The constructs for all test samples based on the cases

Table 3.2: Comparison of Models for reduced search space

Average	**BM**	**EB1**	**EB2**	**EB3**
Average number of teams in the reduced search space	6.07	5.025	7.518	8.79
Reduced search space with ground truth	0.66	0.68	0.78	0.83

discussed in the previous section are computed, and these arguments are used as input to the DeLP argumentation framework. For each test sample the DeLP system is queried to find all possible attackers (culprits) based on the arguments provided. If there is no way to decide between contradicting arguments, these are blocked and thus return no answers. Initially, the search space for each test sample is 19 teams (all except the one being attacked).

After running the queries to return the set of possible culprits, the average search space across all target teams is 6.07 teams. This is a significant reduction in search space across all target teams; to gauge how much the reduced search space can aid an analyst in predicting the actual culprit, a metric is computed that checks if the reduced search space contains the ground truth (actual culprit). For all the target teams, the ground truth is present on average in almost 66% of the samples with reduced search space. For some teams like *more smoked leet chicken* (T-13) and *raon_ASRT (whois)* (T-17) the average reduced search space is as low as 1.82 and 2.9 teams, with high ground truth fractions of 0.69 and 0.63, respectively.

Analysis is then performed on the reduced search space to derive a culprit. The experimental setup is similar to the one described earlier; the only difference this time is instead of having a 19 team search space as in chapter 2, the machine learning approach is allowed to make a prediction from the reduced search space only; a random forest is used for learning, which has been shown to have the best performance on this data as reported in chapter 2.

The accuracy achieved after running Random forest without applying the argumentation-based techniques, as reported in chapter 2, is 0.37. This was the best performing approach

using standard machine learning techniques. The baseline model achieves an average accuracy of 0.5, already a significantly improvement.

Extended Baseline Argumentation Model I (EB1)

We used the same experimental setup discussed in the baseline argumentation model. EB1 performs slightly better than the baseline model with an average accuracy of 0.53 vs. 0.50, and significantly better than the machine learning model without argumentation (accuracy of 0.37). The improvement in performance is due to the fact that we have a larger fraction of reduced search spaces with ground truth present in it; also, the search space reduced from on average 6.07 teams to 5.025 teams, thus having a better chance at finding the right solution.

Extended Baseline Argumentation Model II (EB2)

Again, we used the same experimental setup to evaluate this model; we also report the fraction that the ground truth is present in the reduced search space to give an intuition regarding the performance of the prediction model. In Table 3.3, we can see that EB2 obtained an average accuracy of 62%. The addition of teams based on time rules not only benefits detection of previously unseen deceptive teams, but it also helps in predicting attackers for previously unseen attacks. The major reason for the jump in performance is that in many queries with unseen deceptive teams, the time rules proposed in the baseline model block all deceptive teams from being the culprit, leading to an empty set of culprits. The new set of time rules proposed in EB2 adds similar-behaving teams to this set based on time information; the learning algorithm is then able to predict the right one from this set. Accuracies for each target team are reported in Table 3.4.

Extended Baseline Argumentation Model III (EB3)

We evaluate the last extended baseline model (EB3) using the same experimental setup. In Table 3.3, EB3 performs the best with an average prediction accuracy of 64.5%—Table 3.4 gives the performance for each target team. The conditional time rules benefit 7 out of 20 teams by retaining the actual culprit in the search space, which is then selected by the machine learning model. The retention is not always beneficial, since in 9 cases the accuracy does not change from EB2 to EB3. On the other hand, in 4 cases the performance degrades. The reason for the improvement in overall accuracy is due to the retention of the preferred attacker set for target teams that contain the actual attacker. This is observed in the fact that for EB3 the fraction of ground truth present in the reduced search space increases to 83% as compared to 78% for EB2 (see Table 3.3). Also, the average reduced search space increases from 7.518 teams to 8.79 teams, as more teams are retained due to the conditional time rules.

Table 3.3: Average accuracy comparison of proposed models

Average	**BM**	**EB1**	**EB2**	**EB3**
Reduced search space with ground truth	0.66	0.68	0.78	0.83
Accuracy	0.50	0.53	0.62	0.645

3.5.2 Rule relevance discussion

We now study and discuss the relevance of different rules introduced to address the three test cases, namely non-deceptive, deceptive, and previously unseen attacks.

Case 1: Non-deceptive attacks. For these attacks, the search space only has one team, which is chosen as the culprit (see Fig. 3.5). We define a simple rule that adds the only team that used that exploit as the likely culprit. On average, the non-deceptive attacks make up

Table 3.4: Results Summary

Team	**ML (chapter 2)**	**BM**	**EB1**	**EB2**	**EB3**
T-1	0.45	0.51	0.52	**0.60**	0.58
T-2	0.22	0.45	0.38	0.43	**0.50**
T-3	0.30	0.40	0.47	0.66	**0.68**
T-4	0.26	0.44	0.42	0.44	**0.48**
T-5	0.26	0.45	0.45	0.56	**0.60**
T-6	0.5	0.49	0.55	**0.70**	**0.70**
T-7	0.45	0.53	0.56	**0.66**	0.64
T-8	0.42	0.61	0.58	**0.74**	**0.74**
T-9	0.41	0.50	0.53	**0.76**	**0.76**
T-10	0.30	0.42	0.41	0.41	**0.46**
T-11	0.37	0.44	0.5	**0.73**	0.71
T-12	0.24	0.43	0.36	0.52	**0.56**
T-13	0.35	0.63	0.64	**0.75**	**0.75**
T-14	0.42	0.52	0.53	**0.67**	**0.67**
T-15	0.30	0.38	0.55	0.64	**0.67**
T-16	0.43	0.48	0.55	**0.65**	**0.65**
T-17	0.42	0.58	0.58	**0.68**	**0.68**
T-18	0.48	0.50	0.52	**0.65**	0.63
T-19	0.41	0.51	0.56	**0.68**	**0.68**
T-20	0.48	0.51	0.64	**0.71**	**0.71**

almost 5% of the total test attacks for each target team. On the other hand, the accuracy of choosing the attacker on average for each target team is 67%. Thus, the simple rule engineered for the non-deceptive case works well in determining the attacker for 67% of the time. The misclassified 33% make up a very small portion of the total misclassified test attacks. The reason for misclassification is that the non-deceptive attack *becomes deceptive in the test set* when a new team starts using the exploit. It is a difficult case for attacker selection, since it requires the framework to predict if a particular test case will become deceptive in the future based on the evidence from the knowledge base. No new rules are introduced for the non-deceptive case for any of the proposed extended models.

Case 2: Deceptive attacks. Initially, for deceptive attacks the search space was comprised of all the teams that had used the exploit in the knowledge base (see Fig. 3.6). The deceptive attacks (including deceptive replayed attacks) make up to almost 80% of the test attacks for each target team. Below we discuss the rules introduced for deceptive attacks for each of the proposed models, and how they affect the accuracy of the solutions.

- **BM:** The baseline model introduces time rules that eliminate teams from the culprit set if the time difference since the last attack does not fall in a certain time bucket (see Fig. 3.8). With these time rules, the average accuracy for each target team for the deceptive attacks is 54%. Misclassification is attributed to two main factors: first, the defined time rules are not able to capture the ground truth (i.e., they eliminate the actual culprit from the culprit set); second, the classification model not able to make the right prediction.

- **EB1:** In this baseline model, rules were introduced to address the misclassification for previously unseen attacks, without any new rules added for deceptive attacks; hence, the classification performance is the same as for BM.

- **EB2:** In some cases, the time rules proposed in the BM result in the elimination

of all the teams in the culprit set, thus returning an empty set for the classification model to choose from. Also, since the test attack is already classified as deceptive, teams not using the exploit in the knowledge base might adopt it in the test set. To address both these issues, we introduced additional time rules (see Fig. 3.10) that add similar-behaving teams to the culprit set. In this case the similar behavior represents teams that have similar deceptive attack times. The new time rules solve the issue of the empty culprit set, at the same time adding teams that might adopt the deceptive behavior in the test set. This leads to a significant performance improvement for the deceptive test attacks, boosting the accuracy from 54% to 64%. The performance boost indicates that new teams often adopt deceptive behavior in the test set.

- **EB3:** The time rules proposed above either eliminate teams from the culprit set or they add to it. The new model proposed here introduces conditional time rules where the elimination of teams is subject to whether they belong to the preferred attacker set or not for the particular target team (see Fig. 3.11). It is the only proposed model where contradiction is resolved by looking at the ordering of rules, deciding which rule has priority. In this case the preferred attacker set is given preference, as teams often target teams that they feel are easy to exploit since the team is not quick enough to patch discovered vulnerabilities. This is observed in the knowledge base for each of the target teams. Introduction of these new rules improves the accuracy of deceptive attacks slightly to just under 65%, indicating that rule ordering has some impact.

Case 3: Previously unseen attacks. The attacks not observed in the knowledge base are termed are referred to as previously unseen attacks, and they comprise almost 15% of the test attacks for each target team. The rules introduced for these attacks are either the ones introduced for the deceptive or the non-deceptive case, depending on the computed similar exploits for a given unseen test attack. So any performance improvement due to the

introduction of additional deceptive rules will benefit the unseen attacks as well. We do a similar comparison of the models proposed for unseen attacks as done for the deceptive ones.

- **BM:** In the baseline model, for the unseen attacks we compute the three most similar exploits from the knowledge base, and then categorize the exploit as deceptive or non-deceptive; rules are computed accordingly. The average accuracy for the unseen attack for this model is 30%, so unseen attacks are often misclassified. The main reasons are that the rules are not able to capture the actual attacking team; also, the rule of computing similar exploits assumes that similar exploits will be authored by the same team, which is not true in many cases. Hence, an expansion of the rules for previously unseen attacks is required.

- **EB1:** Two rules are proposed for previously unseen attacks in EB1 (see Fig. 3.9). The first rule computes the number of similar exploits to be used based on similarity threshold. Hence, the number of similar exploits varies for each unseen attack and is not fixed to 3 as in the case of BM. The second rule is designed on the basis that not all teams have similar technical capabilities. Some teams design new exploits throughout the competition, while others monitor the network and capture and use exploits designed by other teams most of the time. Therefore, when a new exploit is encountered in the test set, it is very likely that it comes from a team that has a history of designing new exploits. We thus introduce a rule that computes the three newest exploit designers for a given target team and adds them to the set of culprits. These two rules significantly boost the previously unseen attack performance from 30% to 44%, indicating their relevance.

- **EB2:** Here, time rules are introduced to add similar-behaving teams to the culprit set. These rules had a significant performance boost for the deceptive attacks. Since

previously unseen attacks can also be deceptive, a similar performance improvement was observed. The average accuracy of EB2 increases from 44% to 52% for previously unseen attacks.

- **EB3:** The performance for EB3 has a trend similar to EB2's. The introduced conditional time rules benefit deceptive attacks, improving the average accuracy for previously unseen attacks to 55%.

Table 3.5 summarizes the results for the three test cases for all the proposed models. It can be observed that time rules proposed in EB2 have the most significant performance boost for deceptive attacks; on the other hand, the threshold-based rules and the rules to compute the teams responsible for designing the most unique exploits have the most significant performance boost for previously unseen attacks.

Table 3.5: Comparison of Models for the three test cases

Average Accuracy	**BM**	**EB1**	**EB2**	**EB3**
Non-deceptive attacks	0.67	0.67	0.67	0.67
Deceptive attacks	0.54	0.54	0.64	0.65
Unseen attacks	0.30	0.44	0.52	0.55

3.6 Related Work

This chapter builds upon our own recent application of argumentation to cyber attribution [56] and our preliminary study on applying machine learning models for cyber attribution in the presence of deception as reported in chapter 2. In chapter 2, only machine learning techniques were leveraged on the CTF data to identify the attacker—we will now provide a summary of the results obtained. The experiment was performed as follows: the dataset was divided according to the target team, building 20 subsets, and all the attacks were

then sorted according to time. The first 90% of the attacks were reserved for training, and the remaining 10% for testing. The byte and instruction histograms were used as features to train and test the model; models constructed using a random forest classifier performed the best, with an average accuracy of 0.37. Most of the misclassified samples tend to be deceptive attacks and their replicates. When using machine learning approaches it is difficult to map the reasons why a particular attacker was chosen, especially in cases of deception where multiple attackers were associated with the same attack. Knowing the explanations that supported a particular decision would greatly aid the analyst in making better decisions dealing with uncertainty. To address this issue, in this chapter we described how we can derive arguments based on the latent variables computed from the training data, given an attack for attribution. This chapter proposes a framework, in which an argumentation model is used to reason about deception and integrated with the machine learning model for attacker prediction. This system achieved an accuracy of 62%, significantly improving the performance for deceptive attacks.

In other literature, currently cyber-attribution is limited to identifying machines [14] as opposed to the hacker or their affiliation to a group or a state. An example of such a technical attribution approach is WOMBAT [34], where a clustering technique is used to group attacks to common IP sources. A method that combines information from different sources was proposed by Walls [132], who considered forensic information from diverse sources but did not account for inconsistency or uncertainty due to deception. A less rigorous mathematical approach, known as the Q model [102], was proposed recently; the model answers queries from an analyst, and by combining these answers the analyst attributes an attack to a party. Unfortunately, there are no experimental evaluations of its effectiveness. Similarly, other work [126] combines cluster analysis with a component for multi-criteria decision analysis and studied an implementation of this approach using honeypot data – again, this approach lacks any ground truth of the actual hacker or hacking group.

Argumentation has been used for cyber reasoning [7] by leveraging arguments to deal with incomplete and contradictory data, allowing to derive big-picture conclusions to keep systems secure and online in case of an attack. This is a different application than the one we are addressing. Using argumentation to support human decisions was presented in [117]. Here, the authors discuss how user trust in the evidence influences decision making; demonstrating the hypotheses in a user-study [107]. Concurrently, a formal logical framework for reasoning about cyber-attribution has been devised [56, 114]; it explores multiple competing hypotheses based on the evidence for and against a particular attacker to help analysts decide on an attribution, providing a map of the reasoning that led to the decision. While that work inspired this chapter, it did not contain an empirical evaluation.

Adversarial machine learning is an emerging field of study. It uses effective machine learning techniques to identify or defend against opponents. Understanding the limits of adversaries' knowledge and capabilities is crucial for coming up with countermeasures, as discussed in [53]. Here the authors propose models to study these limitations to come up with evasion techniques. On the contrary, Lowd and Meek [74] explore the problem from an adversary's point of view. They propose strategies that an adversary can use to reverse engineer a classifier so that attacks are undetected by the classifier. They use a real world application in spam filtering to demonstrate their method, which they call adversarial classifier evasion. In a spam filtering setting, an example of such a technique is replacing feature words that raise a red flag with their synonyms to evade detection. This feature cross substitution technique is discussed in [69], where the authors offer a simple heuristic method based on mixed-integer linear programming with constraint generation to make the classifier robust to cross substitution techniques. There is also research that looks at modeling the interaction between the learner (adversary) and the classifier in terms of a competition using Stackelberg games [20, 19]. Most adversarial machine learning applications deal with modeling classifiers to be robust against evasive techniques in real world applications

like malware detection and spam filtering. On the contrary, our proposed system does not model the adversary but instead analyzes the evidence to derive arguments for and against a particular team being the attacker; so, in our system the adversary has no knowledge of arguments as in the case of adversarial machine learning.

3.7 Summary

In this chapter we demonstrated how leveraging Defeasible Logic Programming (DeLP) in an argumentation-based framework, can be leveraged to improve cyber-attribution decisions. This is done by building DeLP programs based on real-world data; this approach affords a reduction of the set of potential culprits and thus greater accuracy when using a classifier for cyber attribution. We thus proposed a hybrid system that integrates argumentation with a learning model to make decisions.

Chapter 4

DETERMINATION OF ADVERSARIAL INTENT

4.1 Introduction

Identifying the tasks [1] a given piece of malware was designed to perform (e.g. logging keystrokes, recording video, establishing remote access, etc.) is a difficult and time consuming task that is largely human-driven in practice [116]. The complexity of this task increases substantially when you consider that malware is constantly evolving, and that how each malware instance is classified may be different based on each cyber-security expert's own particular background. Automated solutions for this problem are highly attractive as they can significantly reduce the time it takes to conduct remediation in the aftermath of a cyber-attack.

Earlier work has sought to classify malware by similar "families", something which has been explored as a supervised classification problem [8, 60, 61]. However, differences over determining "ground truth" for malware families (i.e. Symantec and McAfee cluster malware into families differently) and the tendency for automated approaches to only succeed at "easy to classify" samples [71, 98] are two primary drawbacks of malware family classification. More recently, there has been work on directly inferring the tasks a malware was designed to perform [54]. This approach leverages static malware analysis (i.e. analysis of the malware sample conducted without execution, such as decompilation) and a comparison with a crowd-source database of code snippets using a proprietary machine leaning approach. However, a key shortcoming of the static method is that it is of limited value when the malware authors encrypt part of their code – as we saw with the infamous Gauss malware [59]. This

[1] We use the term task and intent interchangeably throughout the work.

work builds upon recent developments in the application of cognitive models to intelligence analysis tasks [65] and our own preliminary studies on applying cognitive models to identify the tasks a piece of malware was designed to perform [64, 125]. Specifically, in this chapter, we report

- Experimental results illustrating consistent and significant performance improvements (in terms of precision, recall, and F1) of the instance-based cognitive model approach when compared with various standard machine learning approaches (including SVM, logistic regression and random forests) for two different sandboxes and for three different datasets.

- Experimental results showing a consistent and significant performance improvement of the instance-based cognitive model and several other machine learning approaches when compared to the current state-of-the-art commercial technology (based on static analysis).

- Experiments where we study cases where the malware samples are mutated, encrypted, and use different carriers - providing key insights into how our approach will cope with operational difficulties.

- Experimental results illustrating that a cognitively-inspired intermediate step of inferring probability distribution over malware families provides improved performance over the machine learning and rule-based cognitive model (though no significant change to the instance-based cognitive model).

- We also provide run-time comparisons of the experiments and discussing the cognitive models in terms of parameter selection and time complexity analyses. We also explore the concept of predicting hacker intentions on a host machine in real time.

4.2 Technical Preliminaries

Throughout this chapter, we shall assume that we have a set of malware samples that comprise a historical corpus (which we shall denote $\mathcal{M}$) and each sample $i \in \mathcal{M}$ is associated with a set of tasks (denoted *tasks*(i)) and a set of attributes (denoted *attribs*(i)). Attributes are essentially binary features associated with a piece of malware that we can observe using dynamic and/or static analysis while the tasks - which tell us the higher-level purpose of the malware - must be determined by a human reviewing the results of such analysis. As $\mathcal{M}$ comprises our historical knowledge, we assume that for each $i \in \mathcal{M}$ both *tasks*(i) and *attribs*(i) are known. For a new piece of malware, we assume that we only know the attributes. We also note that throughout the chapter, we will use the notation $|\cdot|$ to denote the size of a given set. In Tables 4.1 and 4.2 provide examples of the attributes and tasks based on the malware samples from the Mandiant APT1 dataset (created from samples available at [78], see also [77]). For instance *hasDynAttrib* looks at the behavior section of the analysis report and extracts all the activity of the malware on the host machine. The attribute *usesDll* enumerates all the libraries that were used by the malware on the host machine. The file activity and the registry activity is captured by *fileAct* and *regAct*. Finally all the processes initiated and terminated by the malware are captured by *proAct*. There is not a fixed number of any of these attributes for a given malware. The number of attributes depends on the analysis report generated from the sandbox. A full description of this dataset is presented in Section 4.5.

Throughout the chapter, we will also often consider malware families, using the symbol $\mathcal{F}$ to denote the set of all families. Each malware sample will belong to exactly one malware family, and all malware samples belonging to a given family will have the same set of tasks. Hence, we shall also treat each element of $\mathcal{F}$ as a subset of $\mathcal{M}$.

Table 4.1: Attributes extracted through automated malware analysis

Attribute	**Intuition**
usesDll(X)	Malware uses a library X
regAct(K)	Malware conducts an activity in the registry, modifying key K.
fileAct(X)	Malware conducts an activity on certain file X
proAct	Malware initiates or terminates a process

Table 4.2: Sample of malware tasks

Task	**Intuition**
beacon	Beacons back to the adversary's system
enumFiles	Designed to enumerate files on the target
serviceManip	Manipulates services running on the target
takeScreenShots	Takes screen shots
upload	Designed to upload files from the target

4.3 Cognitively-Inspired Inference

While human inference has memory and attentional limitations, their cognitive processes are powerful, where adaptive heuristic strategies are adopted to accomplish the tasks under strong time constraints using limited means. An advantage of using a cognitive model to describe inferential processes is that the underling architecture provides the benefits of human-inspired inference while allowing for more flexibility over constraints such as human working memory. We believe that there is a valid use of cognitive architectures for artificial intelligence that makes use of basic cognitive mechanisms while not necessarily making use of all constraints of the architecture. Reitter & Lebiere (2010) introduced a modeling methodology called accountable modeling that recognizes that not every aspect of a cognitive model is reflected in measurable performance. In that case, it is arguably better

to specifically state which aspects of the model are not constrained by data, and rather than mock up those aspects in plausible but impossible to validate manner, simply treat them as unmodeled processes. This approach results in simpler models with a clear link between mechanisms used and results accounted for, rather than being obscured by complex but irrelevant machinery. For instance, while the models described in this chapter use activation dynamics well-justified against human behavioral and neural data to account for features such as temporal discounting, we do not directly model working memory constraints to allow for more features of malware and more instances to be present in memory.

4.3.1 ACT-R Based Approaches

We propose two models built using the mechanisms of the ACT-R (Adaptive Control of Thought-Rational) cognitive architecture [5]. These models leverage the work on applying this architecture to intelligence analysis problems [65]. In particular, we look to leverage our recently-introduced instance-based (ACTR-IB) and rule-based (ACTR-R) models [64, 125]. Previous research has argued that the ability of instance-based learning in complex dynamic situations making it appropriate for sensemaking [45]. On the other hand the rule-based learning is a more compact representation of associating samples in memory with their respective families. In this section, we review some of the major concepts of the ACT-R framework that are relevant to these models and provide a description of both approaches.

We leveraged features of the declarative memory and production system of the ACT-R architecture to complete malware task identification. In ACT-R, recall from declarative memory (c.f., identification, for our purposes) depends on three main components: activation strengthening (i.e., the base-level activation of an element), associative (i.e., spreading) activation, and inter-element similarity (i.e., partial matching). These three values are summed together to represent an items total activation. When a recall is requested from memory, the item with the highest total activation is retrieved.

Declarative Knowledge. Declarative knowledge is represented formally in terms of *chunks*. Chunks have an explicit type, and consist of an ordered list of slot-value pairs of information. Chunks are retrieved from declarative memory by an activation process, and chunks are each associated with an *activation strength* which in turn is used to compute a *retrieval probability*. In this chapter, chunks will typically correspond to a malware family. In the version of ACTR-IB where we do not represent families explicitly, the chunks correspond with samples in the training data.

For a given chunk i, the activation strength A_i is computed as,

$$A_i = B_i + S_i + P_i \tag{4.1}$$

where, B_i is the base-level activation, S_i is the spreading activation, and P_i is the partial matching score. We describe each of these in more detail as follows.

Base-Level Activation (B_i): Technically, base-level for chunk i reflects both the frequency and recency of samples in memory, even though we are not using recency here but it could easily be applicable to weigh samples toward the more recent ones. More important, base-level is set to the log of the prior probability (i.e., the fraction of samples associated with the chunk) in ACTR-R; for instance-based (ACTR-IB), we set it to a base level constant β_i.

Spreading Activation (S_i): Spreading activation is a measure of the uniqueness of the attributes between a test sample i and a sample j in memory. The spread of activation to sample i is computed by the summing the strengths of association between sample j and the attributes of the current sample i being considered. To compute the spreading activation we compute the *fan* of attribute a (i.e., the number of samples in memory with attribute a) for each attribute. The strength of association is computed differently in both approaches and, in some cognitive model implementations, is weighted (as is done in ACTR-R of this chapter).

Partial Matching (P_i): A partial matching mechanism computes the similarity between two

samples. In this work, it is only relevant to the instance-based approach. Given a test sample j, its similarity with a sample i in memory is computed as a product of the mismatch penalty (mp, a parameter of the system) and the degree of mismatch M_{ji}. We define the value of M_{ji} to be between 0 and -1; 0 indicates complete match while -1 complete mismatch.

As common with models based on the ACT-R framework, we shall discard chunks whose activation strength is below a certain threshold (denoted τ). Once the activation strength, A_i, is computed for a given chunk, we can then calculate the activation probability, Pr_i. This is the probability that the cognitive model will recall that chunk and is computed using the Boltzmann(softmax) equation [121], which we provide below.

$$Pr_i = \frac{(e^{\frac{A_i}{s}})}{\sum_j (e^{\frac{A_j}{s}})} \tag{4.2}$$

Here, e is the base of the natural logarithm and s is momentary noise inducing stochasticity by simulating background neural activation (this is also a parameter of the system).

4.3.2 ACT-R Instance-Based Model

The instance based model is an iterative learning method that reflects the cognitive process of accumulating experiences (in this case the knowledge base of training samples) and using them to predict the tasks for unseen test samples. Each malware instance associates a set of attributes of that malware with its family. When a new malware sample is encountered, the activation strength of that sample with each sample in memory is computed using Equation 4.1. The spreading activation is a measure of the uniqueness of the attributes between a test sample i and a sample j in memory. To compute the spreading activation we compute the fan for each attribute a ($fan(a)$ finds all instances in memory with the attribute a) of the test sample i. The Partial matching is computed as explained above. The degree of mismatch is computed as the intersection between the attribute vector of the given malware and each sample in memory normalized using the Euclidean distance between the two vectors. The

retrieval probability of each sample j in memory with respect to the test sample i is then computed using Equation 4.2. This generates a probability distribution over families. The tasks are then determined by summing up the probability of the families associated with that task with an appropriately set threshold (we set that threshold at 0.5(indicates that the model should be more than 50% confident before a task is predicted for a test malware sample)). Algorithm 1 shows the pseudo code for the instance-based model.

ALGORITHM 1: ACT-R Instance-based Learning

INPUT: New malware sample i, historical malware corpus $\mathcal{M}$.

OUTPUT: Set of tasks associated with sample i.

for query malware sample i **do**

for all j in $\mathcal{M}$ **do**

$B_j = \beta_j$

$P_j = mp \times \frac{|attribs(i) \cap attribs(j)|}{\sqrt{|attribs(i)| \times |attribs(j)|}}$

for $a \in attribs(i)$ **do**

if $a \in attribs(j)$ **then**

s_{ij} += $log(\frac{|\mathcal{M}|}{|fan(a)|})$

else

s_{ij} += $log(\frac{1}{|\mathcal{M}|})$

end if

end for

$S_j = \sum_j \frac{s_{ij}}{|attribs(i)|}$

Calculate A_j as per Equation 4.1

end for

Calculate Pr_j as per Equation 4.2

$p_f = \sum_{j \in f \text{ s.t. } A_j \geq \tau} Pr_j$

$t_p = \{t \in T | p_f \geq 0.5\}$

end for

Time Complexity of Instance-based Model: The Instance based model has no explicit training phase, so there are no training costs associated with it. For a given test sample the model computes the activation function for each sample in the knowledge base. Hence the time complexity increases linearly with the knowledge base. Let n be the number of the samples in the knowledge base and m is the number of attributes associated with the test sample, then the time complexity can be given as $O(nm)$ for each test sample, as we expect m to be relative small ($n >> m$), the relationship is linear in n.

4.3.3 ACT-R Rule-Based Model

In this version of ACT-R model we classify the samples based on simple rules computed during the training phase. Given a malware training sample with its set of attributes a, along with the ground truth family value, we compute a pair of conditional probabilities $p(a|f)$ and $p(a|\neg f)$ for an attribute in a piece of malware belonging (or not belonging) to family f. These probabilistic rules (conditional probabilities) are used to set the strength of association of the attribute with a family ($s_{a,f}$). The strength of association is weighted by the source activation w to avoid retrieval failures for rule-based models. We use empirically determined Bayesian priors $p(f)$ to set the base-level of each family as opposed to using a constant base-level for instance based. Only two components of the activation Equation 1 are used, namely the base-level and the spreading activation. Given the attributes for current malware , we calculate the probability of the sample belonging to each family according to Equation 2, generating a probability distribution over families.The task are then determined in a similar way to that of instance-based model. Algorithm 2 shows the pseudo code for the rule-based model.

Time Complexity of Rule-based Model: For Rule-based model computing the rules for each attribute in the knowledge base significantly adds to the computation time. Let n be the number of samples in the training set, m be the number of attributes in the new piece

ALGORITHM 2: ACT-R Rule-based Learning

INPUT: New malware sample i, historical malware corpus $\mathcal{M}$.

OUTPUT: Set of tasks associated with new sample i.

TRAINING:

Let $X = \bigcup_{j \in \mathcal{M}} attrib(j)$

for all a in X **do**

Compute the set of rules $p(a|f)$ and $p(a|\neg f)$

(where $p(a|f) = \frac{|\{i \in \mathcal{M} \cap f | a \in attrib(i)\}|}{|f|}$

and $p(a|\neg f) = \frac{|\{i \in \mathcal{M} - f | a \in attrib(i)\}|}{|\mathcal{M}| - |f|}$)

end for

TESTING:

for all $f \in \mathcal{F}$ **do**

$B_f = log(p(f))$ (where $p(f) = \frac{|f|}{|\mathcal{M}|}$)

for all $a \in attrib(i)$ **do**

$s_{a,f}$ = $log(\frac{p(a|f)}{p(a|\neg f)})$; S_f =+ $\frac{w \times s_{a,f}}{|attribs(i)|}$

end for

A_f = B_f + S_f

end for

Calculate p_f as per Equation 4.2

$t_p = \{t \in T | p_f \geq 0.5\}$

of malware, and m^* be the cardinality of $\bigcup_{j \in \mathcal{M}} attrib(j)$. The resulting time complexity for training is then $O(m^* n)$ for training, which is significant as we observed $m^* >> m$ in our study. While this is expensive, we note that for testing an individual malware sample, the time complexity is less than the testing phase for the instance based $O(|\mathcal{F}|m)$ - though the instance based model requires no explicit training phase (which dominates the time complexity of the training phase for the rule-based approach).

4.3.4 Model Parameter Settings

The two proposed models leverage separate components of the activation function. Table 4.3 provides a list of parameters used for both the ACT-R models - we use standard ACT-R parameters that have been estimated from a wide range of previous ACT-R modeling studies from other domains [137] and which are also suggested in the ACT-R reference manual [15].

The intuition behind these parameters is as follows. The parameter s injects stochastic noise in the model. It is used to compute the variance of the noise distribution and to compute the retrieval probability of each sample in memory. The mismatch penalty parameter mp is an architectural parameter that is constant across samples, but it multiplies the similarity between the test sample and the samples in knowledge base. Thus, with a large value it penalizes the mismatch samples more. It typically trades off against the value of the noise s in a signal-to-noise ratio manner: larger values of mp lead to more consistent retrieval of the closest matching sample whereas larger values of s leads to more common retrieval of poorer matching samples.The activation threshold τ determines which samples will be retrieved from memory to make task prediction decisions. The base level constant β is used to avoid retrieval failures which might be caused due to high activation threshold. The source activation w is assigned to each retrieval to avoid retrieval failures for rule-based models.

Table 4.3: Parameters for the Cognitive models

Model	**Parameters**
Instance Based Learning	β = 20 (base-level constant) s = 0.1 (stochastic noise parameter) τ = -10 (activation threshold) mp = 20 (mismatch penalty)
Rule Based learning	s = 0.1 (stochastic noise parameter) w = 16 (source activation)

4.4 Experimental Setup

4.4.1 Baseline Approaches

We compare the proposed cognitive models against a variety of baseline approaches - one commercial package and five standard machine learning techniques discussed in Section 2.3. For the machine learning techniques, we generate a probability distribution over families and return the set of tasks associated with a probability of 0.5 or greater while the commercial software was used as intended by the manufacturer. Parameters for all baseline approaches were set in a manner to provide the best performance.

Commercial Offering: Invencia Cynomix. Cynomix is a malware analysis tool made available to researchers by Invencia industries [54] originally developed under DARPA's Cyber Genome project. It represents the current state-of-the-art in the field of malware capability detection. Cynomix conducts static analysis of the malware sample and uses a proprietary algorithm to compare it to crowd-sourced identified malware components where the functionality is known.

4.4.2 Dynamic Malware Analysis

Dynamic analysis studies a malicious program as it executes on the host machine. It uses tools like debuggers, function call tracers, machine emulators, logic analyzers, and network sniffers to capture the behavior of the program. We use two publicly available malware analysis tools to generate attributes for each malware sample. These tools make use of a sandbox, which is a controlled environment to run malicious software.

Anubis Sandbox. Anubis [55] is an online sandbox which generates an XML formated report for a malware execution in a remote environment. It generates detailed static analysis of the malware but provides less details regarding the behavior of the malware on the host machine. Since it is hosted remotely we cannot modify its settings.

Cuckoo Sandbox. Cuckoo [29] is a standalone sandbox implemented using a dedicated virtual machine and more importantly can be customized to suit our needs. It generates detailed reports for both static as well as behavior analyses by watching and logging the malware while its running on the virtual machine. These behavior analyses prove to be unique indicators (behavior patterns common to a single family) for a given malware for the experiments.

4.4.3 Performance Evaluation

In our tests, we evaluate performance based primarily on four metrics: precision, recall, unbiased F1, and family prediction accuracy. For a given malware sample being tested, precision is the fraction of tasks the algorithm associated with the malware that were actual tasks in the ground truth. Recall, for a piece of malware, is the fraction of ground truth tasks identified by the algorithm. The unbiased F1 is the harmonic mean of precision and recall. In our results, we report the averages for precision, recall, and unbiased F1 for the number of trials performed. Our measure of family accuracy - the fraction of trials where the most

probable family was the ground truth family of the malware in question - is meant to give some insight into how the algorithm performs in the intermediate steps.

4.5 Results

All experiments were run on Intel core-i7 operating at 3.2 GHz with 16 GB RAM. Only one core was used for experiments. Except where explicitly noted, the ACT-R parameters were fixed as per Table 4.3 for all experiments (across all datasets and sandboxes).

4.5.1 Mandiant Dataset

Our first set of experiments uses a dataset based on the T1 cyber espionage group as identified in the popular report by Mandiant Inc [77]. This dataset consisted of 132 real malware samples associated with the Mandiant report that were obtained from the Contagio security professional website [78]. Each malware sample belonged to one of 15 families including BISCUIT, NEWSREELS, GREENCAT and COOKIEBAG. Based on the malware family description [77], we associated a set of tasks with each malware family (that each malware in that family was designed to perform). In total, 30 malware tasks were identified for the given malware samples (see Table 4.2). On average, each family performed 9 tasks.

We compared the four machine learning approaches with the rule-based and instance-based ACT-R models (ACTR-R and ACTR-IB respectively). We also submitted the samples to the Cynomix tool for automatic detection of capabilities. These detected capabilities were then manually mapped to the tasks from the Mandiant report. Precision and recall values were computed for the inferred adversarial tasks. On average the machine learning approaches predicted 9 tasks per sample, ACTR-R predicted 9 tasks per sample and ACTR-IB predicted 10 tasks. On the other hand Cynomix was able to detect on average only 4 tasks.

Leave one out Cross-Validation (LOOCV).

In leave one out cross validation, for n malware samples, train on $n-1$ samples and test on the remaining one. This procedure was repeated for all samples and the results were averaged. We performed this experiment using both sandboxes and compared the results (see Table 4.4).

Table 4.4: Performance comparison of Anubis and Cuckoo Sandbox

Method	**Anubis (F1)**	**Cuckoo (F1)**
DT	0.80	0.80
NB	0.71	0.74
LOG-REG	0.82	0.85
SVM	0.86	0.90
RF	0.89	0.89
ACTR-R	0.85	0.88
ACTR-IB	**0.93**	**0.96**

Method	**Anubis (Family)**	**Cuckoo (Family)**
DT	0.59	0.63
NB	0.30	0.40
LOG-REG	0.65	0.84
SVM	**0.85**	0.86
RF	0.82	0.86
ACTR-R	0.73	0.89
ACTR-IB	0.81	**0.93**

The average F1 increases by 0.03 when we use the attributes generated by the Cuckoo sandbox instead of Anubis. The statistical significance results are as follows: for ACTR-IB (t (132) = 1.94, p = 0.05), ACTR-R (t (132) = 1.39, p = 0.16), RF (t (132) = 0.56, p = 0.57), SVM (t (132) = 1.95, p = 0.05), LOG-REG (t (132) = 1.82, p = 0.07), NB (t (132) = 1.79, p = 0.08) and DT (t (132) = 0.83, p = 0.4). But the significant improvement was in the family prediction values with ACTR-IB improving by 0.12 from 0.81 to 0.93 (t (132) = 3.86, p $<$ .001) and ACTR-R by 0.15 from 0.72 to 0.87 (t (132) = 3.78, p $<$ .001) outperforming all other methods. Since having behavior analysis helps in better task prediction as seen from the comparison experiment, we use cuckoo sandbox for rest of our experiments.

Fig. 4.1 compares the performance of the five best performing methods from Table 1 and compares it with the Cynomix tool of Invincea industries. ACTR-IB outperformed

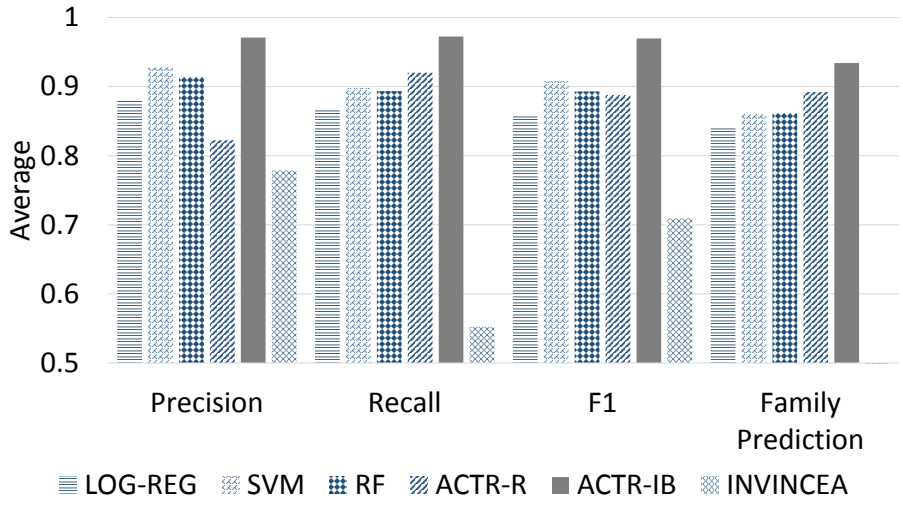


Fig. 4.1: Average Precision, Recall, F1 and Family prediction comparisons using cuckoo sandbox for LOG-REG, RF, SVM, ACTR-R, ACTR-IB and INVINCEA.

LOG-REG, SVM, RF and ACTR-R; average F1 = 0.97 vs 0.85 ($t(132) = 7.85$, $p < .001$), 0.9 ($t(132) = 4.7$, $p < .001$), 0.89 ($t(132) = 5.45$, $p < .001$) and 0.88 ($t(132) = 5.2$, $p < .001$) respectively. Both the proposed cognitive models and machine learning techniques significantly outperformed the Cynomix tool in detecting the capabilities (tasks).

These three approaches (LOG-REG, SVM, RF) were also evaluated with respect to predicting the correct family (before the tasks were determined). ACTR-IB outperformed LOG-REG, SVM, RF and ACTR-R; average family prediction = 0.93 vs 0.84 ($t(132) = 3.22$, $p < .001$), 0.86 ($t(132) = 3.13$, $p < .001$), 0.86 ($t(132) = 3.13$, $p < .001$) and 0.89 ($t(132) = 2.13$, $p = .03$) respectively.

Task Prediction without inferring families:

In the proposed models we infer the malware family first and then predict the tasks associated with that family. However, differences over "ground truth" for malware families in the cyber-security community calls for a direct inference of tasks without dependence on family

prediction. In this section we adapt the models to predict tasks directly without inferring the family.

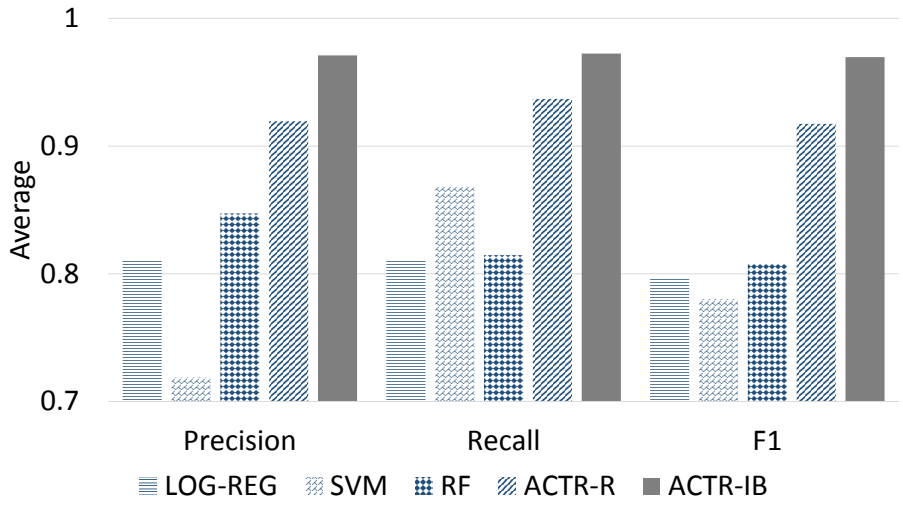


Fig. 4.2: Average Precision, Recall, and F1 comparisons for LOG-REG, RF, SVM, ACTR-R and ACTR-IB for Mandiant without inferring families.

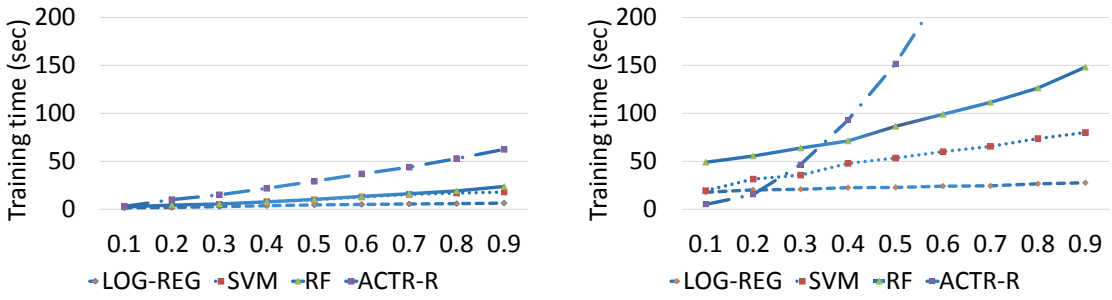


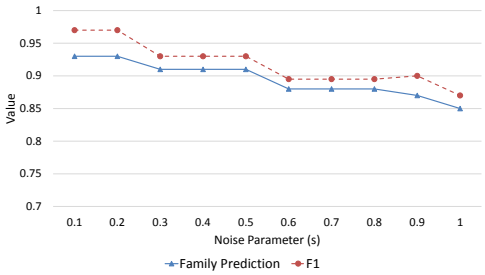
Fig. 4.3: Training time for LOG-REG, SVM, RF and ACTR-R with(left) / without(right) inferring families.

Fig. 4.2 shows the performance of the cognitive and machine learning models without inferring the families.There is no difference in the performance of ACTR-IB and ACTR-R approaches as compared to Fig.2 where we use families. On the other hand direct task

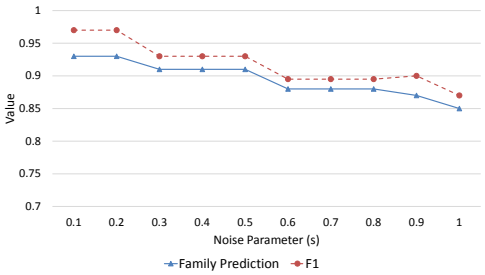
prediction reduces the F1 measure of machine learning techniques on average by almost 0.1. This is due to the fact that, now instead of having a single classifier for each family we have multiple classifiers for each task that a malware sample is designed to perform. This not only degrades the performance but also adds to the training time for these methods (including the ACT-R rule-based approach). We compare the training time with increase in training data for task prediction with/without inferring families. Inferring families first reduces the training time (see Fig. 4.3 (a)). On the other hand predicting tasks directly significantly increases the training time for the machine learning methods along with the rule-based ACT-R approach (Fig. 4.3 (b)). Due to the issues with respect to performance and training time, we consider inferring families first for the rest of the experiments. An important point to note is this has no effect on the Instance-based model for both performance and computation time.

Parameter Exploration:

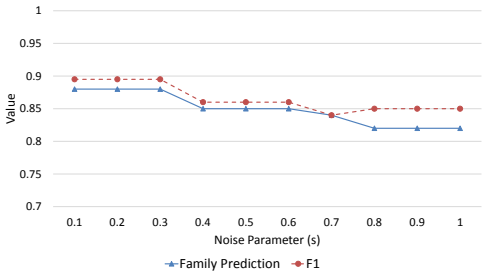
We now discuss two system parameters that control the performance of the ACT-R instance based model namely the stochastic noise parameter (s) and the activation threshold (τ). We use the Mandiant dataset to perform this evaluation. The parameter s takes values between 0.1 and 1 (typical values range from 0.1 to 0.3). The value of the activation threshold depends on the application. Fig. 4.4 shows the variation of family prediction accuracy and F1 score with respect to different noise parameter values and for different activation thresholds. The parameter s is used to compute the variance of the noise distribution and retrieval probability of sample in memory. Larger value of s triggers the retrieval of poor matching samples, which leads to lower family prediction and F1 scores. As seen in the Fig. 4.4, as the value of s increases the performance decreases. On the other hand, the activation threshold dictates how many closely matched samples will be retrieved from memory. For high values of τ the performance decreases as many fewer samples are retrieved. For lower values of τ we end up retrieving almost all the samples in the training data, hence the performance does not



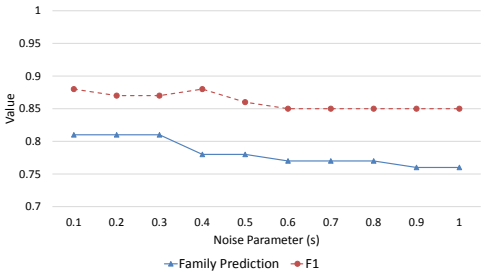
(a) τ = -20



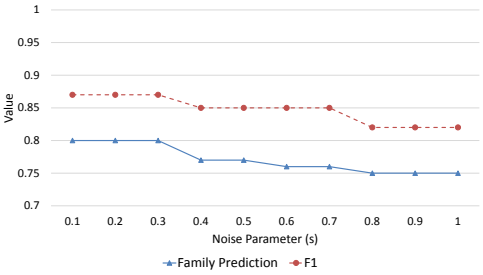
(b) τ = -10



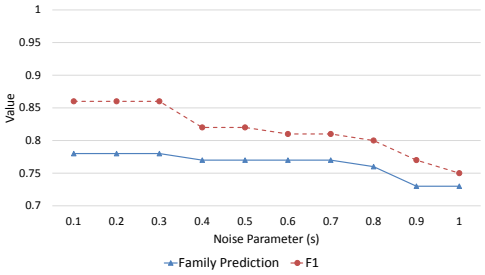
(c) τ = 0



(d) τ = 5



(e) τ = 10



(f) τ = 15

Fig. 4.4: Family prediction and F1 value for different threshold and noise parameters values.

decrease as τ decreases, but it adds to the computational cost of retrieving high number of samples which is not desirable. We get the best performance for τ = -10 and s = 0.1. Even s = 0.2 is almost as good as 0.1 providing some advantages in terms of stochasticity ensuring robustness

We keep the base-level constant (β) and mismatch penalty (mp) values constant. As explained earlier the base-level constant trades off directly against the retrieval threshold, and the mismatch penalty against the activation noise, respectively, so it makes sense to vary only one of the pair.

4.5.2 GVDG Dataset

GVDG is a malware generation tool designed for the study of computer threats [47]. It is capable of generating the following malware threats:

- File-virus
- Key-Logger
- Trojan-Extortionist
- USB-Worm
- Web Money-Trojan

Fig. 4.5 shows the GVDG user interface used for the generation of malware samples. We can select the carrier type and the tasks that we want the malware sample to perform on the host machine. The tasks are represented as payloads, while carrier is a functional template which can be modified to execute the tasks desired by the user on the host system. In generating datasets with GVDG, we specify families based on sets of malware with the same tasks. Whether or not a family consists of malware with the same carrier depends on the experiment. Further, GVDG also has an option to increase "mutation" or variance among the samples. We perform experiments analyzing the performance of the proposed

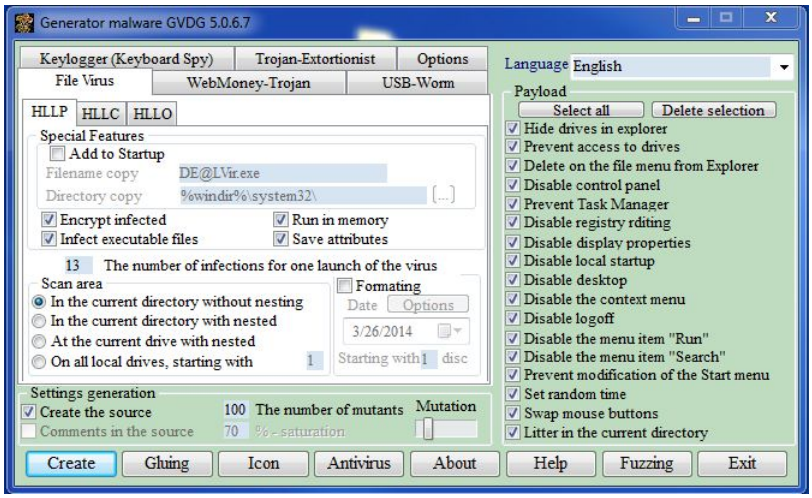


Fig. 4.5: GVDG User Interface

methods when the generated samples belong to different carrier and same carrier types, as well as when the samples are encrypted and mutated making task prediction difficult. In all the experiments we consider 60% of the data for training and 40% for testing. The results are averaged across 10 trials. The Cynomix tool from Invencia was unable to detect any tasks for the GVDG dataset, primarily due to its inability to find public source documents referencing GVDG samples and also unable to generalize from similar samples.

Different Carriers:

In this experiment, we generated 1000 samples for each carrier type with low mutation. On average each carrier type performs 7 tasks(payloads). Hence each carrier represents one family for this experiment. Both random forest and ACTR-IB model were able to predict the tasks and family with F1 measure of 1.0 outperforming LOG-REG 1 vs 0.91 , SVM 1 vs 0.95 and ACTR-R 1 vs 0.95. All results are statistical significant with (t (1998) $\geq$ 8.93, p $<$ .001)(Fig. 4.6). Also for family prediction ACTR-IB and RF outperformed LOG-REG 1 vs 0.92, SVM 1 vs 0.92 and ACTR-R 1 vs 0.95 (t (1998) $\geq$ 8.93, $<$ .001).

These results are not surprising given that different carrier(family) types have high

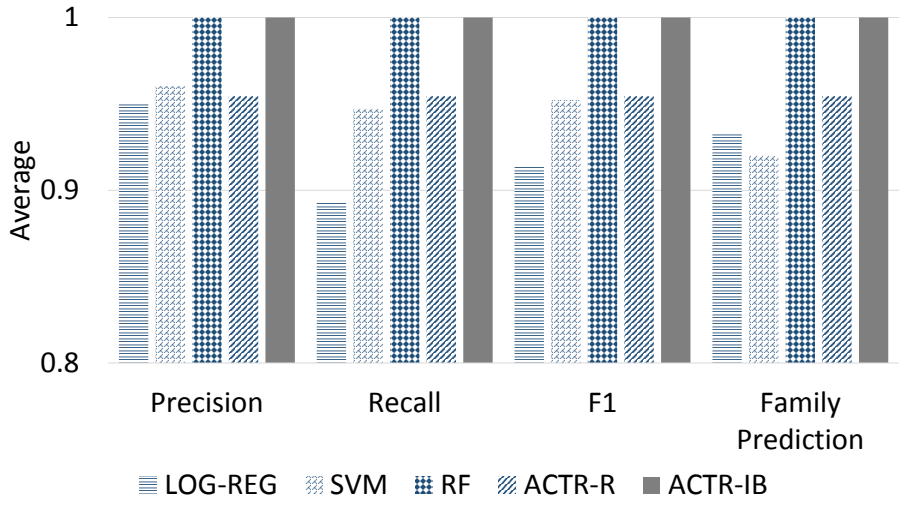


Fig. 4.6: Average Precision, Recall, F1 and Family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for different carrier samples.

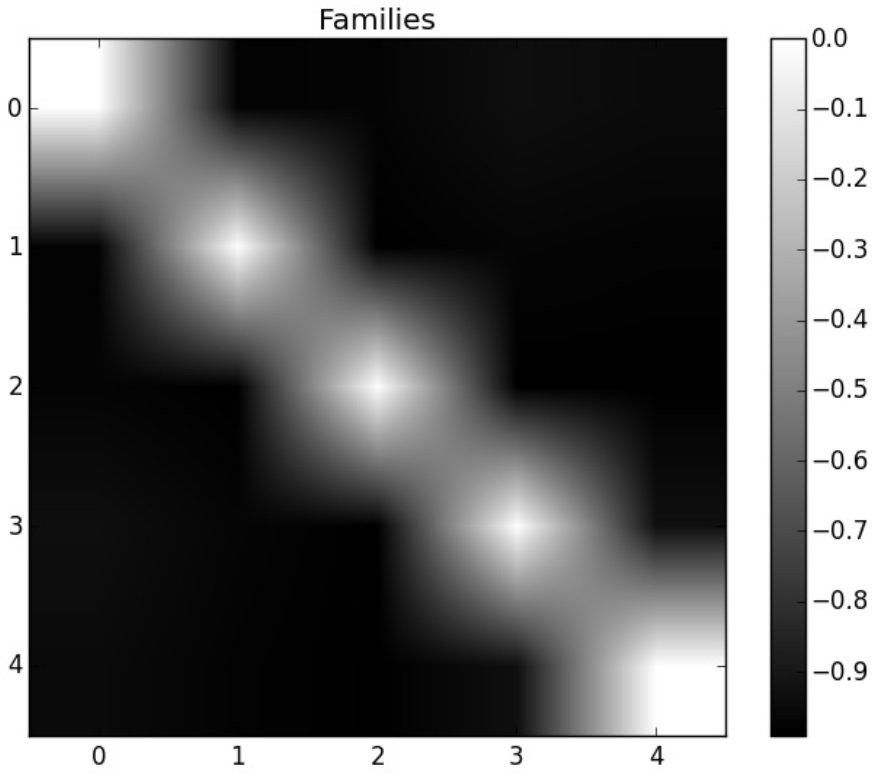


Fig. 4.7: Similarity matrix for 5 different carriers .

dissimilarity between them. Also, samples belonging to the same carrier have on average 60% of similar attributes. Fig. 4.7 shows the similarity between the carrier types. The similarity between families is calculated in the same way as ACTR-IB partial matching with 0 indicating complete match while -1 complete mismatch.

Different Carriers-Mutation:

For this case, we generate the same samples as in the previous experiment but with maximum mutation between samples belonging to the same carrier. We generated 1000 samples for each carrier with maximum mutation. In this case ACTR-IB had an average F1 of 1 outperforming LOG-REG 1 vs 0.83, SVM 1 vs 0.88 , RF 1 vs 0.96 and ACTR-R 1 vs 0.92 (t (1998) ≥ 7, p $< .001$)(Fig. 4.8). Also for family prediction ACTR-IB outperformed LOG-REG 1 vs 0.85, SVM 1 vs 0.88 , RF 1 vs 0.95 and ACTR-R 1 vs 0.92 (t (1998) ≥ 7, p $< .001$).

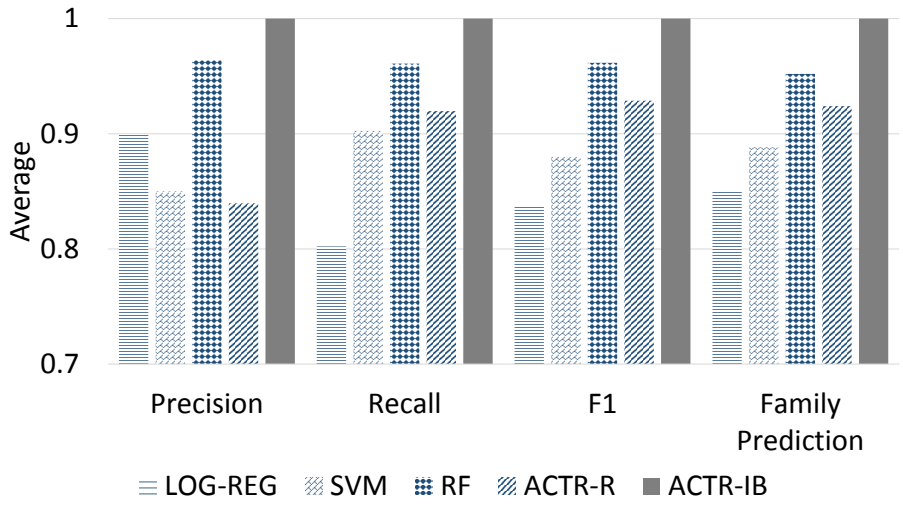


Fig. 4.8: Average Precision, Recall, F1 and Family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for different carrier mutated samples.

High mutation induces high variance between samples associated with the same carrier making the classification task difficult. High mutation samples belonging to same carrier

have only 20% of common attributes as compared to 60% for low mutation.

Less Training data:

In order to see how the cognitive models perform with less training data, we repeated the

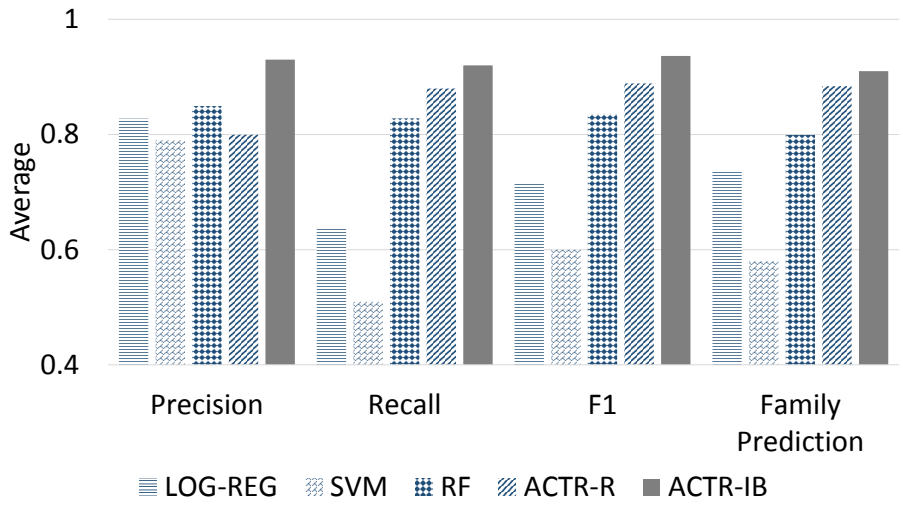


Fig. 4.9: Average Precision, Recall, F1 and Family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for less training data.

different-carrier mutation experiment with 10% of the training data selected uniformly at random (300 samples). Even with less training data ACTR-IB had an average F1 of 0.93 outperforming LOG-REG 0.93 vs 0.71, SVM 0.93 vs 0.6, RF 0.93 vs 0.83 and ACTR-R 0.93 vs 0.88 ($t(1998) \geq 2.89$, $p \leq .001$), see Fig. 4.9. Also for family prediction ACTR-IB outperformed LOG-REG 0.91 vs 0.73 ($t(1998) = 19.3$, $p < .001$), SVM 0.91 vs 0.58, RF 0.91 vs 0.79 and ACTR-R 0.91 vs 0.88 ($t(1998) \geq 2.05$, $p \leq 0.04$).

Different Carriers: Low-High Mutation:

For this case, we consider the low mutation samples as training data and the high mutation samples as testing. Fig. 4.10 shows the comparison results. ACTR-IB had an average F1

of 0.96 outperforming LOG-REG 0.96 vs 0.83, SVM 0.96 vs 0.92, RF 0.96 vs 0.93 and ACTR-R 0.96 vs 0.88 (t (2498) $\geq$ 15.7, p $<$ .001), see Fig. 4.10. Also for family prediction ACTR-IB outperformed LOG-REG 0.96 vs 0.81, SVM 0.96 vs 0.92, RF 0.96 vs 0.94 and ACTR-R 0.96 vs 0.88 (t (2498) $\geq$ 7, p $<$ .001).

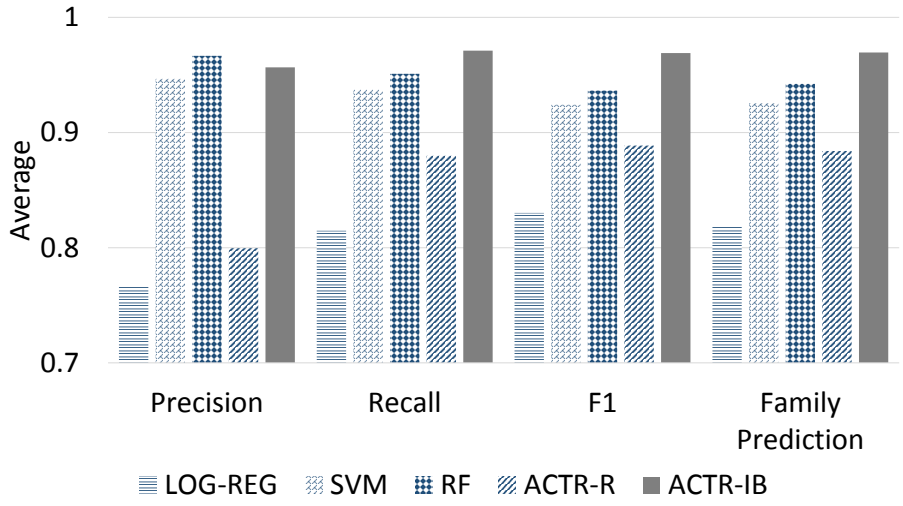


Fig. 4.10: Average Precision, Recall, F1 and Family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for low-high mutated samples.

Leave one carrier out cross-validation:

To see how the models generalize to unseen malware family(carrier), we performed a leave-one-carrier-out comparison, where we test the models against one previously unseen malware carrier. ACTR-IB performs better or on par with all other baseline approaches for all the carriers. It clearly outperforms all the approaches in recalling most of the actual tasks (40%) (see Fig. 4.11). ACTR-IB has shown to generalize for unseen malware families [64]. This case is difficult given the fact that the test family is not represented during training, hence task prediction depends on associating the test family with the training families that perform similar tasks.

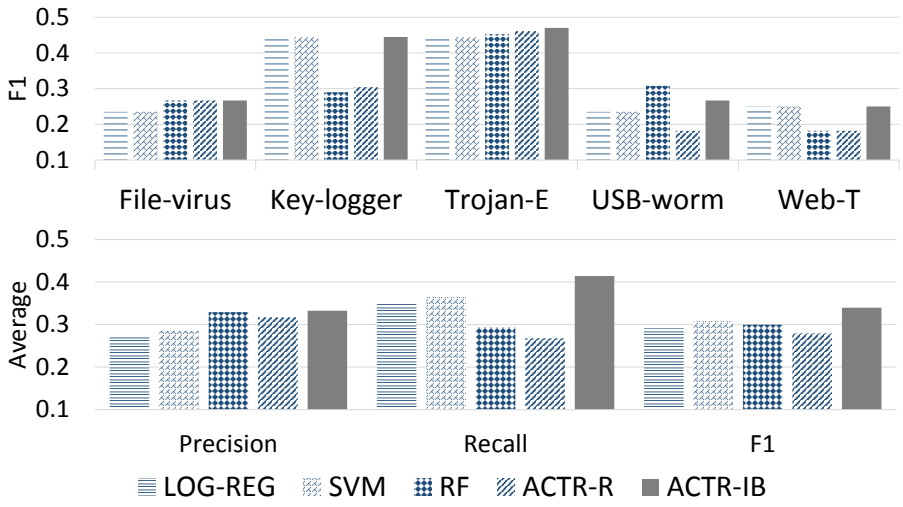


Fig. 4.11: Average F1 values for 5 malware carriers (above) and the average precision, recall and F1 across all carriers (below) for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for Leave-one-carrier-out.

Same Carrier:

As seen in the previous experiments, different carrier types makes the task easier because of less similarity between them. We now test the performance, on same carrier type performing exactly one task. Since there are 17 tasks in the GVDG tool, we generate 100 samples for each task for carrier type File-virus. In this experiment each task represents one family. Thus in total we have 1700 samples. We do the 60-40 split experiment. From Fig. 4.12 ACTR-IB had an average F1 of 0.95 outperforming LOG-REG 0.95 vs 0.84, SVM 0.95 vs 0.87, RF 0.95 vs 0.90 and ACTR-R 0.95 vs 0.92 ($t(678) \geq 1.52$, $p \leq 0.13$). Since each family performs exactly one task the family prediction is similar to F1. Using the same carrier for each payload makes the task difficult as can be seen from the similarity matrix for the 17 payloads(Fig. 4.13).

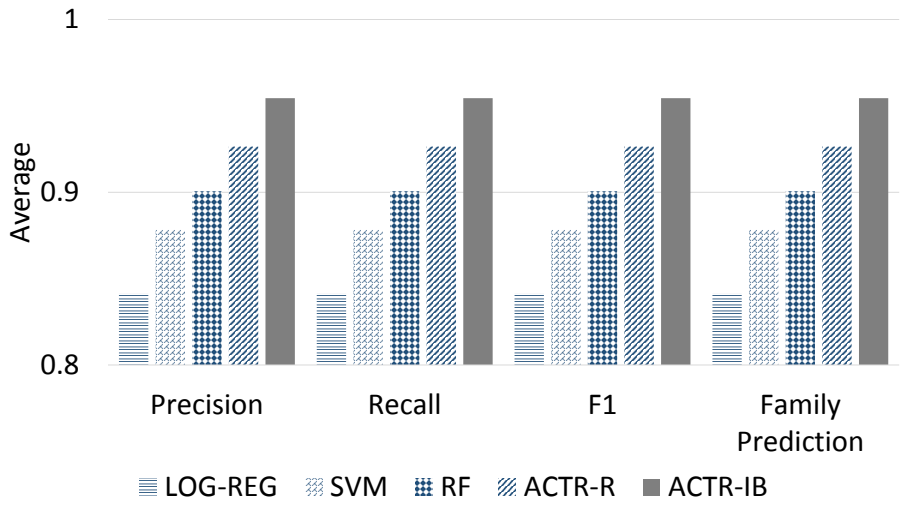


Fig. 4.12: Average Precision, Recall, F1 and Family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for unencrypted same carrier samples.

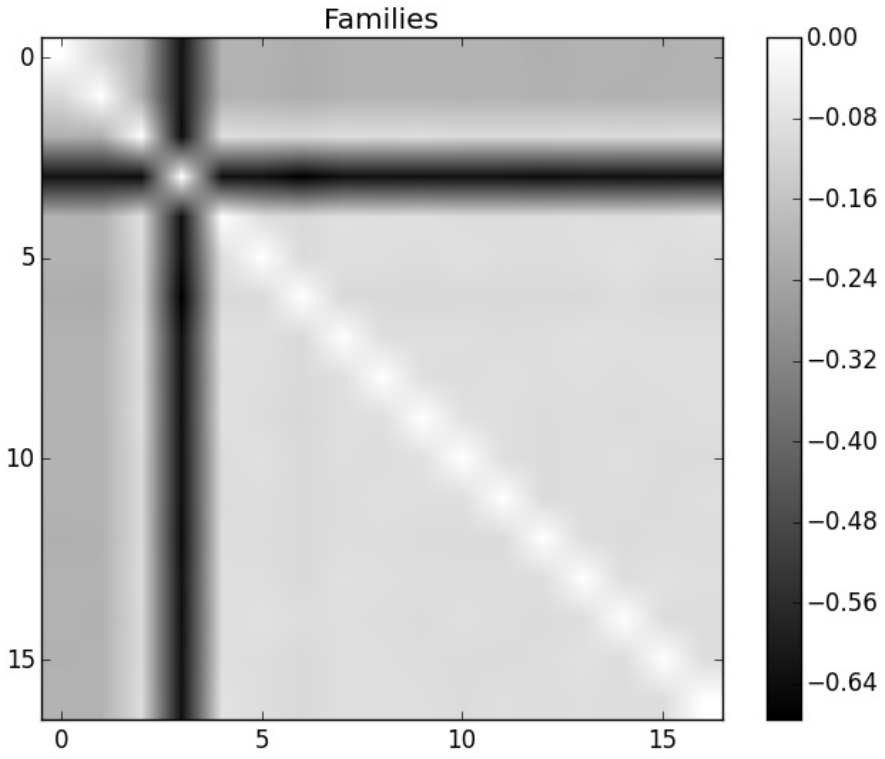


Fig. 4.13: Similarity matrix for 17 versions of the same carrier .

Same Carrier-Encryption:

The GVDG tool provides the option for encrypting the malware samples for the File-virus carrier type. We use this option to generate 100 encrypted malware samples for each task(payload) and use them as test data with the unencrypted versions from the same carrier experiment as training samples. From Fig. 4.14 ACTR-IB had an average F1 of 0.9 outperforming LOG-REG 0.9 vs 0.8, SVM 0.9 vs 0.8, RF 0.9 vs 0.74 and ACTR-R 0.9 vs 0.88 ($t(1698) \geq 2.36$, $p \leq 0.02$). Encrypting malware samples morphs the task during execution making it difficult to detect during analysis. Hence the drop in performance as compared to non-encrypted samples.We note that SVM performs better than RF likely because it looks to maximize generalization.

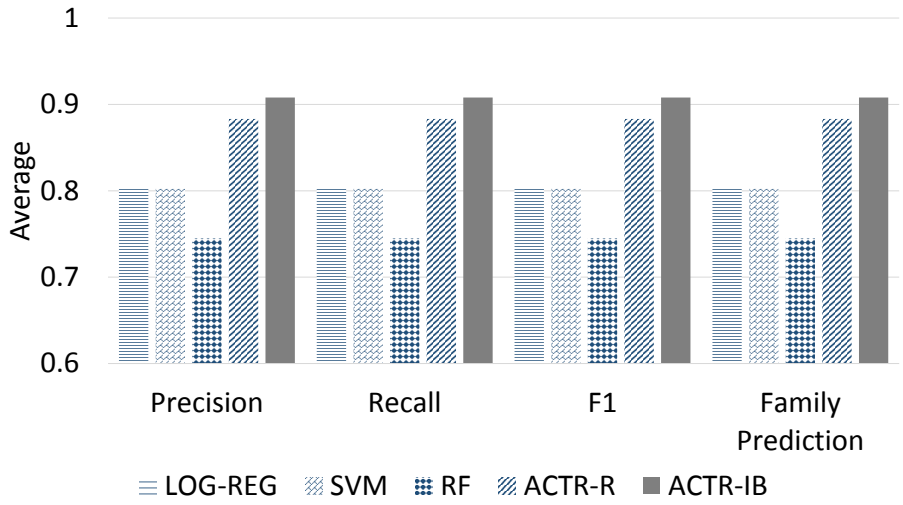


Fig. 4.14: Average Precision, Recall, F1 and Family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for encrypted same carrier samples.

Runtime analysis:

Table 4.5 shows the classifier run times for the experiments. Machine learning techniques are faster but have large training times, which increase almost linearly with the size of the

knowledge base. Hence updating the knowledge base is computationally expensive for these methods, as it has to re-estimate the parameters every time. The same notion holds true for ACTR-R, since computing the rules during training phase is expensive as can be seen from the large training times. ACTR-IB on the other hand has no explicit training phase, so the only time cost is during testing. In fact ACTR-IB is faster than SVM and RF for same/encrypted carrier experiments.

Scaling of Instance-based model:

Finally to conclude the GVDG experiments, we run ACTR-IB on a combination of all the above variations of dataset to highlight the space requirements for the learning model. The dataset comprises of 5 different carriers with low/high mutation (10,000 samples) and same carrier encrypted/non-encrypted (3400 samples). Based on the tasks they perform we have in total 22 families represented by 13,400 samples. The analysis reports generated by cuckoo take up 4 gigabytes of disk space for the samples. We significantly reduce the size to 600 megabytes by parsing the analysis reports and extracting attributes. We set aside 10% of the samples for testing (1340) and iteratively add 10% of the remaining data for training. Table 4.6 gives a summary of the average F1 measure and testing time for ACTR-IB. The results are averaged across 10 trials. There is a steady increase in performance till we reach 40% of the training data, after that the F1 measure remains almost constant. This experiment clearly indicates the ability of the ACTR-IB to learn from small amount of representation from each family, significantly reducing the size of the knowledge base required for training. We are also looking into techniques to reduce the time requirements of instance-based learning algorithm (e.g., Andrew Moore explored efficient tree-based storage). There are also techniques for reducing space requirements, [110] merged training instances in the ACT-R-Gammon model and obtained considerable space savings at little performance cost.

Table 4.5: Classifier run times

Experiment	**Model**	**Train(sec)**	**Test(sec)**
different carriers	LOG-REG	202	7
	SVM	250	50
	RF	280	30
	ACTR-R	6443	143
	ACTR-IB	-	453
mutated carriers	LOG-REG	214	18
	SVM	260	63
	RF	303	85
	ACTR-R	7223	185
	ACTR-IB	-	465
same carriers	LOG-REG	152	4.22
	SVM	270	38
	RF	290	55
	ACTR-R	4339	120
	ACTR-IB	-	205
encrypted carriers	LOG-REG	180	15
	SVM	300	80
	RF	353	110
	ACTR-R	6103	180
	ACTR-IB	-	365

4.5.3 MetaSploit

MetaSploit is a popular penetration testing tool used by security professionals to identify flaws in the security systems by creating attack vectors to exploit those flaws [95]. Penetra-

Table 4.6: Summary of ACTR-IB results

Fraction of training data	**F1 measure**	**Test time(sec)**
0.1	0.77	418
0.2	0.82	839
0.3	0.90	1252
0.4	0.97	1676
0.5	0.97	2100
0.6	0.97	2525
0.7	0.97	2956
0.8	0.98	3368
0.9	0.98	3787
1.0	0.98	4213

tion testing may also be defined as the methods an attacker would employ to gain access to security systems. Hence identifying the tasks the exploit was designed to perform is important to counter the exploit.

For this experiment we generate exploits that attacks windows operating systems. Each exploit has a set of tasks associated with it. The tasks include setting up tcp & udp back-door connections, adding unauthorized users to the system, modifying root privileges, download executables and execute them on the local machine, prevent writing of data to disk, deleting system folders, copying sensitive information etc. We generated 4 exploit families with 100 samples each performing on average 4 tasks. We induced mutation between samples belonging to the same family making the classification task difficult. We perform a 60-40 split training-testing experiment and average the results across 10 trials. From Fig. 4.15 ACTR-IB had an average F1 of 0.86 outperforming LOG-REG 0.86 vs 0.62, SVM 0.86 vs 0.82, RF 0.86 vs 0.82, ACTR-R 0.86 vs 0.81 and Invencia 0.86 vs 0.8 ($t(158) \geq 1.94$, $p \leq$

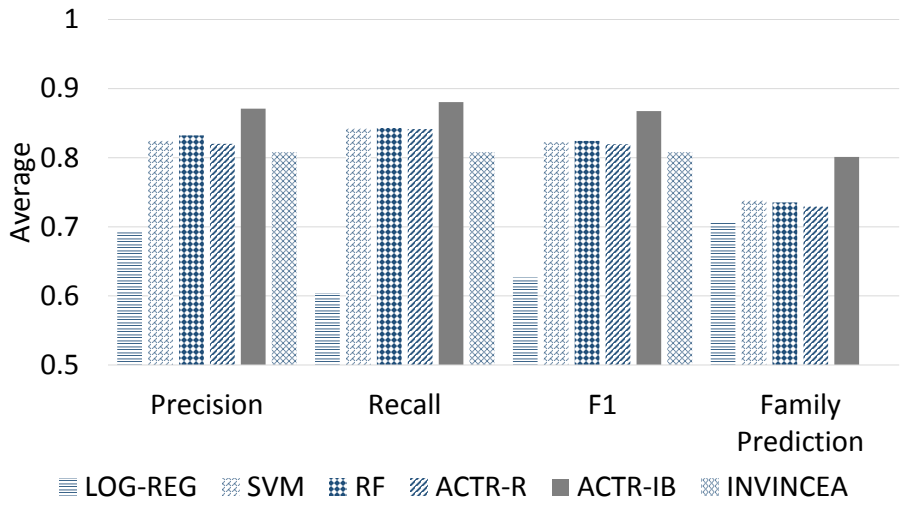


Fig. 4.15: Average Precision, Recall, F1 and Family prediction comparisons for LOG-REG, SVM, RF, ACTR-R and ACTR-IB for MetaSploit samples.

0.05). Also for family prediction ACTR-IB outperformed LOG-REG 0.8 vs 0.7, SVM 0.8 vs 0.72, RF 0.8 vs 0.72 and ACTR-R 0.8 vs 0.71 ($t(158) \geq 2.53$, $p \leq 0.01$).

4.5.4 Task Prediction from Hacker activities

In all the experiments discussed so far, the tasks associated with a given piece of malware are predefined and do not change with time. In this section , we try to map the tasks that a hacker is trying to achieve from the activities it performs on a compromised system. For the entire experiment only one malware is used whose sole purpose is to create a tcp backdoor connection to let the hacker have access to the system. We evaluate the test samples only using ACTR-IB and not other machine learning methods. The goal of this experiment is to demonstrate how the system can deal with real time hacker activities on a compromised system. It demonstrates the capability of the proposed system to capture hacker behavior.

The experimental setup is as follows. We keep the Cuckoo sandbox running on the system by executing the malware. This will create a connection between the hacker and

the system. Once the hacker gains control of the machine, he can perform operations in order to achieve his objectives. We treat these objectives as the tasks that the hacker wants to complete on the system. Once these tasks are completed, Cuckoo generates an analysis report detailing the behavioral analysis of the hacker. However, these analytics are too detailed for instance file and registry execution commands and do not provide a clear picture of the main tasks of the hacker on the machine. Hence, traditionally, this will often require an expert security analyst to go through large analysis results to determine the task which is often time consuming. But instead we can feed the analysis report to the ACTR-IB model to get a prediction of the hacker tasks. For this experiment we use the MetaSploit dataset discussed in Section 4.5.3 as the knowledge base for the instance based approach. For the test set we generate samples in real time with hackers trying to achieve their goals (tasks) on the compromised system. Note, this test also illustrates how well our model generalizes, as we are identifying hacker behavior using historical data that was not generated by the hacker - or even a human in this case. We consider two hackers, who are given a list of the payloads (tasks) to complete from the list mentioned in Section 4.5.3. They always perform a fraction of the tasks assigned to them at a given time instance and then the model is tested on predicting these tasks.

We generate 10 such attacks , 5 from each hacker. Each attack consists of achieving 5 tasks on average. We note that for each of the test sample the malware used is the same. ACTR-IB results are presented in Table 4.7. The results are averaged for each hacker across test samples. Table 4.7 shows the actual and predicted tasks for Hacker-1 for 5 different attack instances. The results for Hacker-2 were analogous.

4.6 Related Work

Identification of malicious software. The identification of whether or not binary is malicious [38, 123] is an important related, yet distinct problem from what we study in this

Table 4.7: Summary of ACTR-IB results

Subject	**Average Precision**	**Average Recall**	**Average F1**
Hacker-1	0.8	0.85	0.83
Hacker-2	0.85	0.85	0.85

chapter and can be regarded as a "first step" in the analysis of suspicious binaries in the aftermath of a cyber-attack. However, we note that as many pieces of malware are designed to perform multiple tasks, that successful identification of a binary as malicious does not mean that the identification of its associated tasks will be a byproduct of the result - and hence this is normally the case, which has led to some of the other related work described in this section.

Malware family classification. There is a wealth of existing work on malware family identification [8, 60, 61, 70, 97, 48, 109, 6, 28]. The intuition here is that by identifying the family of a given piece of malware, an analyst can then more easily determine what it was designed to do based on previously studied samples from the same family. However, malware family classification has suffered from two primary drawbacks: (1) disagreement about malware family ground truth as different analysts (e.g. Symantec and McAfee) cluster malware into families differently; and (2) previous work has shown that some of these approaches mainly succeed in "easy to classify" samples [71, 98], where "easy to classify" is a family that is agreed upon by multiple malware firms. In this chapter, we infer the specific tasks a piece of malware was designed to carry out. While we do assign malware to a family as a component of our approach, to avoid the two aforementioned issues as the family partition is done so probabilistically and the result ground truth is the focus of our comparison (though we show family prediction results as a side-result). Further, we also describe and evaluation a variant of our instance-based method that does not consider

Table 4.8: Actual and predicted Hacker-1 attacks

Attack Instance	**Actual Tasks**	**Predicted Tasks**
1	setup backdoor connection modify root privileges uninstall program copy files	setup backdoor connection modify root privileges uninstall program delete system files prevent access to drive
2	setup backdoor connection modify root privileges download executables execute files copy files	setup backdoor connection modify root privileges download executables execute files delete files
3	setup backdoor connection modify root privileges add unauthorized users start keylogging uninstall program delete files prevent access to drives	setup backdoor connection modify root privileges add unauthorized users start keylogging unistall program delete files
4	setup backdoor connection add unauthorized users prevent writing data to disk delete files copy files	setup backdoor connection add unauthorized users prevent writing data to disk delete files modifying root privileges prevent access to drives
5	setup backdoor connection download executables execute files start keylogging	setup backdoor connection download executables execute files start keylogging

families and yields a comparable performance to our instance-based method that does consider families.

Malware task identification. With regard to direct inference of malware tasks, the major

related work include the software created by the firm Invincea [54] for which we have included a performance comparison. Additionally, some of the ideas in this chapter were first introduced in [64, 125, 88]. However, that work primarily focused on describing the intuitions behind the cognitive modeling techniques and only included experimental evaluation on two datasets (the Mandiant APT1 and GVDG datasets). The experimental evaluation in this chapter includes additional experiments for the GVDG dataset to consolidate the previous experiments. Also algorithm analysis and parameter exploration is provided for the cognitive models. In addition we introduce a popular penetration tool used by security analyst MetaSploit and present new results on this tool. These experiments evaluate the proposed model thoroughly to pave the way toward deployment for use by cyber-security analysts.

4.7 Summary

In this chapter, we introduced an automated method that combines dynamic malware analysis with cognitive modeling to identify malware tasks. This method obtains excellent precision and recall - often achieving an unbiased F1 score of over 0.9 - in a wide variety of conditions over three different malware sample collections and two different sandbox environments - outperforming a variety of baseline methods.

Chapter 5

DARKWEB AND DEEPWEB MINING FOR CYBERSECURITY THREAT INTELLIGENCE

5.1 Introduction

With the widespread use of technology, cyber-security has become an important issue of concern for both commercial organizations and governments. With the recent incidents of data breach at Home Depot, Target and Sony via malicious softwares, many organizations are looking at proactive techniques to avoid being targeted or minimize the damage of such attacks.

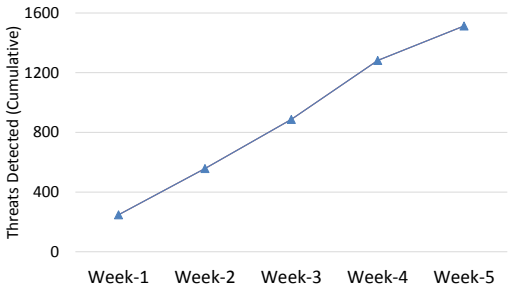


Fig. 5.1: Weekly detection of cyber-threats.

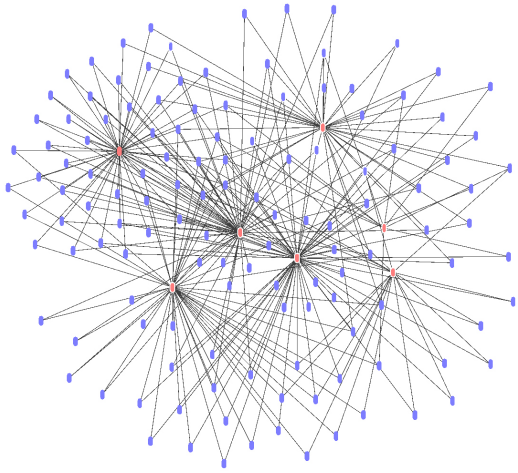


Fig. 5.2: Social network.

Pre-reconnaissance cyber threat intelligence refers to information gathered before a malicious party interacts with the defended computer system. An example demonstrating the importance of cyber threat intelligence is shown in Table 5.1. A Microsoft Windows vulnerability was identified in Feb. 2015. The release of the vulnerability was essentially

Table 5.1: Exploit example.

Timeline	**Event**
February 2015	Microsoft identifies Windows vulnerability MS15-010/CVE 2015-0057 for remote code execution. There was no publicly known exploit at the time the vulnerability was released.
April 2015	An exploit for MS15-010/CVE 2015-0057 was found on a dark-web market on sale for 48 BTC (around $10,000-15,000).
July 2015	FireEye identified that the Dyre Banking Trojan, designed to steal credit card number, actually exploited this vulnerability [1] .

Microsoft warning its customers of a security flaw. Note that at this time, there was no publicly known method to leverage this flaw in a cyber-attack (i.e. an available exploit). However, about a month later an exploit was found to be on sale in darkweb market. It was not until July when FireEye, a major cybersecurity firm, identified that the Dyre Banking Trojan designed to steal credit cards exploited this vulnerability - the first time an exploit was reported. This vignette demonstrates how threat warnings gathered from the darkweb can provide valuable information for security professionals. The average global exposure of the Dyre Banking Trojan was 57.3% along with another banking malware Dridex [2] . It means that nearly 6 out of 10 organizations in the world were affected, and this is a significantly high number on a global level.

In another instance, 17-year-old hacker Sergey Taraspov, from St. Petersburg. along with a small team of hackers, allegedly wrote a piece of malware that targeted point-of-sale (POS) software, and sold it for $2,000 on a Russian forum/marketplace. This malware was, in turn, used by around forty individuals to steal over 110 million American credit card

[2]https://www.fireeye.com/blog/threat-research/2015/06/evolution_of_dridex.html

numbers in the ”Target“- data breach of 2013 [3] .

In this chapter, we examine how such intelligence can be gathered and analyzed from various social platforms on the Internet particularly sites on the darkweb and deepweb. In doing so, we encounter several problems that we addressed with various data mining techniques. At the time of development the system was actively collecting approximately 305 cyber threats each week. Fig. 5.2 shows a subset of a social network built using the collected data. Since then we have transitioned the system to a commercial partner [4] . Fig. 5.1 shows the cumulative trend in threat detection for five weeks at the time of development. Table 5.2 shows the database statistics before transition. It shows the total data collected and the data related to malicious hacking. The vendor and user statistics cited only consider those individuals associated in the discussion or sale of malicious hacking-related material, as identified by the system. The data is collected from three sources on the darkweb/deepweb: markets, forums, and subreddits.

We are providing information to cyber-security professionals to support their strategic cyder-defense planning to address questions such as,

1. *What vendors and users have a presence in multiple darkweb/deepweb markets/ forums?*

2. *What zero-day exploits are being developed by malicious hackers?*

3. *What vulnerabilities do the latest exploits target?*

In this chapter we provide,

- Description of a system for cyber threat intelligence gathering from various social platforms from the Internet such as deepweb and darkweb websites.

[3]https://www.nbcnews.com/news/world/skilled-cheap-russian-hackers-power-american-cybercrime-n22371

[4]Cyber Reconnaissance, Inc. (CRY3CON), https://www.cyr3con.com.

Table 5.2: Current Database Status

Markets	Total Number	27
	Total products	11991
	Hacking related	1573
	Vendors	434
Forums	Total Number	21
	Topics/Posts	23780/162872
	Hacking related	4423/31168
	Users	5491
Subreddits	Total Number	33
	Topics/Posts	3940/19601
	Hacking related	1654/8270

- The implementation and evaluation of learning models to separate relevant information from noise in the data collected from these online platforms.
- A machine learning approach to aid security experts in the discovery of new relevant deepweb and darkweb websites of interest using topic modeling– this reduces the time and cost associated with identifying new deepweb and darkweb sites.
- A series of case studies showcasing various findings relating to malicious hacker behavior resulting from the data collected by our operational system..

5.2 Background

Many of the individuals behind cyber-operations – originating outside of government run labs or military commands – rely on a significant community of hackers. They interact through a variety of online forums (as means to both stay anonymous and to reach geograph-

ically dispersed collaborators). For instance, the distribution of *MegalodonHTTP* Remote Access Trojan (RAT) utilized the amateur black hat platform, HackForum. Five people accused of the malware's creation and/or distribution resided in three European countries, requiring law enforcement to cooperate internationally in pursuit of the malicious hackers' arrest [134]. The international nature of the cyber-domain - the organization of cooperating malicious hackers as well as their international targets - transcends not only territorial executive powers, but illustrates the importance of virtual communication platforms.

Darknet and Deepnet Sites. Widely used for underground communication, "The Onion Router" (Tor) is free software dedicated to protect the privacy of its users by obscuring traffic analysis as a form of network surveillance [36]. The network traffic in Tor is guided through a number of volunteer-operated servers (also called "nodes"). Each node of the network encrypts the information it blindly passes on neither registering where the traffic came from nor where it is headed [36], disallowing any tracking. Effectively, this allows not only for anonymized browsing (the IP-address revealed will only be that of the last node), but also for circumvention of censorship [5] . Here we will use "darkweb" to denote the anonymous communication provided by crypto-networks like "Tor", which stands in contrast to "deepweb" which commonly refers to websites hosted on the open portion of the Internet (the "Clearnet"), but not indexed by search engines [62]. Corporate websites supporting employees and library catalogs are good examples of deepweb presences.

Malicious Hacking. Hacking as a subculture has been the subject of many publications, amongst them Steven Levy's seminal "hackers" [68], which outlines ideological premises which many early computer geeks and programmers shared. The machines comprising the early computers were extensions of the self [130], which might compliment the creative ownership and the demand for free software that permeated Levy's account [68]. The term "hacker" in recent use (and especially in popular media) has become restricted to individuals

[5] See the Tor Project's official website (https://www.torproject.org/)

who seek unauthorized access to computers and computer networks not their own with the purpose to manipulate, steal, log or alter data or structures [49, 118].

Markets. Users advertise and sell their wares on marketplaces. Darknet marketplaces provides a new avenue to gather information about the cyber threat landscape. The marketplaces sell goods and services relating to malicious hacking, drugs, pornography, weapons and software services. Only a small fraction of products (13% in our collected data to date) are related to malicious hacking. Vendors often advertise their products on forums to attract attention towards their goods and services. Marketplaces have wallets to deposit digital currency into, but sometimes forum administrators serve as an escrow service. Products are most often verified before any funds are released to the seller. If a seller is misleading or fails to deliver the appropriate item, they are banned from the site. Similarly, buyers can be banned for not complying with the transaction rules.

Forums. Forums are user-oriented platforms that have the sole purpose of enabling communication. It provides the opportunity for the emergence of a community of like-minded individuals - regardless of their geophysical location. Administrators set up Darkweb forums with communication safety for their members in mind. During registration (though not necessarily with every login) every prospective member has to complete CAPTCHAS (Completely Automated Public Turing test to tell Computers and Humans Apart), answer simple questions, solve puzzles or complete simple arithmetic operation, presumably to prevent automated access. Discussion forums on the Darkweb consist of boards and sub-boards (also called "child-boards") filled with threads concerned with different topics. While structure and organization of Darkweb-hosted forums might be very similar to more familiar web-forums, the topics and concerns of the users vary distinctly. In the English clandestine Darkweb, people interested in cats, steampunk, and the latest conspiracy theories convene, but an abundance of arenas dedicated to child pornography (CP), drugs, and weapons can also be found. Other forums appear to be venues for sharing erotic images – whether

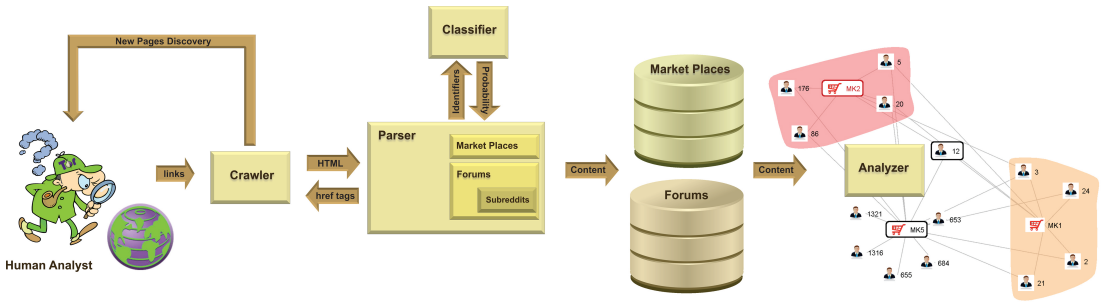


Fig. 5.3: System overview

involving real persons or cartoon characters. Lengthy threads seek information on the reliability of individual vendors and marketplaces in general. Links to other Darkwebsites and information on potentially fraudulent websites are especially useful in the absence of pervasive search engines and can be found on many forums. Forums addressing malicious hackers feature discussions on programming, hacking, and cyber-security. Threads are dedicated to security concerns like privacy and online-safety - topics which plug back into and determine the structures and usage of the platforms.

Subreddits. Subreddits can be considered as a subset of forums. Important information regarding the marketplace environment including reviews of marketplaces, products, and vendors are often discussed on subreddits. These links and sentiments about markets provide insight. For instance, we might learn to predict when popular opinion shifts with respect to a certain market. Subreddits also provide information concerning marketplaces and forums that are newly introduced or old ones that are shutting down. Hence crawling these subreddits could provide good insight into the marketplace and forum environment.

5.3 SYSTEM OVERVIEW

Fig. 5.3 gives the overview of the system. Through search engines and spider services on the Tor network, human analysts were able to find forums and marketplaces populated by malicious hackers. Other platforms were discovered through links posted on forums

either on the Tor-network or on the Clearnet. The system consists of three main modules built independently before integration. The system is currently fully integrated and actively collecting cyber threat intelligence.

Crawler: The crawler is a program designed to traverse the website and retrieve HTML documents. Topic based crawlers have been used for focused crawling where only webpages of interest are retrieved [83, 24]. More recently, focused crawling was employed to collect forum discussions from darkweb [41]. We have designed separate crawlers for different platforms (markets/forums) identified by experts due to the structural difference and access control measures for each platform. In our crawler we address design challenges like accessibility, unresponsive server, repeating links creating a loop etc. to gather information regarding products from markets and discussions on forums.

Parser: We designed a parser to extract specific information from marketplaces (regarding sale of malware/exploits) and hacker forums (discussion regarding services and threats). This well-structured information is stored in a relational database. We maintain two databases, one for marketplaces and the other for forums. Like the crawler, each platform has its own parser. The parser also communicates with the crawler from time to time for collection of temporal data. The parser communicates a list of relevant webpages to the crawler, which are re-crawled to get time-varying data. For markets we collect the following important products fields: $\{$*item_title, item_description, vendor_name, shipping_details, item_reviews, items_sold, CVE, items_left, transaction_details, ratings*$\}$. For forums and subreddits we collect the following fields: $\{$*topic_content, post_content, topic_author, post_author, author_status, reputation, topic_interest*$\}$.

Classifier: Automating the process of classifying a web page as being relevant to the topic of interest greatly expedites data collection. As the crawler traverses darkweb links faster than any human could possibly classify the sites, it is important that the classification portion of the pipeline is able to keep up. By requiring humans to classify each site as relevant or

irrelevant, there is a bottleneck in the classification stage and the throughput of the data gathering pipeline is greatly diminished. We address unique classification challenges. First, to discover new relevant websites using a topic-modeling technique. Second, we employ a machine learning technique using an expert-labeled dataset to detect relevant products and topics from marketplaces and forums respectively. These classifiers are integrated into the parser to filter out products and threads relating to drugs, weapons, etc. not relevant to malicious hacking.

5.4 Evaluation

We address distinct classification problems in this chapter. First, a model that could identify relevant products in darkweb/deepweb marketplaces, relevant topics on forum post containing communication relevant to malicious hacking, and relevant references to marketplaces and forums in subreddits (subreddits are forums in which information relating to forums and marketplaces are discussed - this can be viewed as meta-content). The second problem is identifying new relevant marketplaces/forums. All the problems are binary classification problems with the data sample being relevant or not. We look at both supervised and semi-supervised approaches to address the first classification problem and topic modeling for the second. We now provide an overview of the approaches used and then discuss the classification challenges associated with each problem. For supervised approaches we use standard machine learning techniques discussed in Section 2.3.

5.4.1 Semi-supervised Approaches

Labeling data is expensive and often requires expert knowledge. Semi-supervised approaches work with limited labeled data by leveraging information from unlabeled data. We discuss popular semi-supervised approaches used in this work.

Label propagation (LP). The label propagation approach [139] has been widely used for

semi-supervised classification task [11, 67, 133, 27]. It estimates the label values based on graph Laplacian [9] where the model is represented by a weighted graph $G = (V, E)$, where V indicates the vertices representing the samples , while the edges E are the weights indicating the similarity between points. A subset of these vertices are labeled and these vertices are then used to estimate the labels of the remaining under the assumption that the edges are able to capture the similarity between samples. Hence the performance of these methods depends on the similarity measure used. The most commonly used similarity measures include k-NN and Guassian kernel.

Co-training (CT). The Co-training approach was proposed by Blum and Mitchell [13]. In this approach the feature set is divided into two sets (assumed to be independent), and two classifiers are trained using the limited labeled set denoted by L . These trained classifiers are then used to estimate the labels for the unlabeled points. High confidence label estimates from classifier-1 are added to the labeled set L of classifier-2 and vice versa. For the current setting we set the confidence to 70%. Every time the labeled set L is updated, the classifiers are retrained. This procedure repeats until all of the unlabeled points are labeled. It can be viewed as two classifiers teaching each other. For this approach to work, it is necessary that the two classifiers are uncorrelated and are able to make independent decisions. We implement this idea of using multiple classifiers to gain two different viewpoints of the same data. We implement the co-training algorithm using different classifiers discussed in Section 2.3 to determine the best performing approach.

Latent Dirichlet Allocation (LDA). Additionally, we use LDA, to engineer features in some cases. Latent Dirichlet Allocation is an unsupervised modeling technique. The goal is to infer topics that maximize the likelihood (posteriori probability) of positive webpages. Each webpage has the same set of topics distribution over topics is individually determined for each. The topic proportions are drawn from Dirichlet distribution. These topic distributions can be used as features for classification. The features are limited to the

topics and thus LDA can be used as a dimensionality reduction technique [12].

5.4.2 Experiments: Marketplaces

The recent growth in popularity of darkweb marketplaces provides a new avenue to gather information about the cyber threat landscape. As stated earlier, these marketplaces often sell goods and services that do not relate to malicious hacking, including drugs, pornography, weapons and software services. Only a small fraction of products (13%) are related to malicious hacking. We thus require a model that can separate relevant products from the non-relevant ones. The data collected from marketplaces is noisy and hence not suitable to use directly as input to a learning model. Hence, the raw information undergoes several steps of automated data cleaning. We now discuss the challenges associated with the dataset obtained and the data processing steps taken to address them. We note that similar challenges occur for forum and subreddit data.

Classification Challenges

Text Cleaning. Product title and descriptions on marketplaces often have much text that serves as noise to the classifier (e.g. *****SALE*****). To deal with these instances, we first removed all non-alphanumeric characters from the title and description. This, in tandem with standard stop-word removal, greatly improved classification performance.

Misspellings and Word Variations. Misspellings frequently occur on forums and marketplaces, which is an obstacle for the standard bag-of-words classification approach. Additionally, with the standard bag-of-words approach, variations of words are considered separately (e.g. hacker, hack, hackers, etc.). Word stemming mitigates these issue of word variations, but fails to fix the issue of misspellings. To address this we use character n-gram features. As an example of character n-gram features, consider the word "hacker". If we were using tri-gram character features, the word "hacker" would yield the features "hac", "ack", "cke",

"ker". The benefit of this being that the variations or misspellings of the word in the forms "hack", "hackz", ""hackker", will all have some common features. We found that using character n-grams in the range (3, 7) outperformed word stemming in our experiments.

Large Feature Space. In standard bag-of-words approach, as opposed to the character n-gram approach, the feature matrix gets very large as the number of words increase. Consider the case where the training corpus contains 100,000 unique words and 10,000 documents. The feature matrix then has 100,000 entries for each document, this means that there are 1 billion entries in the feature matrix. As the number of unique words and documents grow, this bloated feature matrix begins to greatly degrade performance. Using n-gram features further increase the already over-sized feature matrix. To address this issue we leveraged the sparse matrix data structure in the scipy [6] library, which leverages the fact that most of these over 1 billion entries will be zero. If a word or n-gram feature is not present in a given document, there is simply no entry for that feature in the sparse matrix. Switching from a dense matrix representation to a sparse matrix representation greatly reduced runtime, as the dense matrix representation was hardly tractable for even a few hundred documents.

Preserving Title Feature Context. As the title and description of the product are disjoint, we found that simply concatenating the description to the product title before extracting features led to sub-optimal classification performance. We believe that by doing a simple concatenation, we were losing important contextual information. There were features that should be interpreted differently should they appear in the title versus the description. Initially, we used two separate classifiers: one for the title and one for the description. With this construction, when an unknown product was being classified, we would pass the title to the title classifier and the description to the description classifier. If either classifier returned a positive classification, we would assign the product a positive classification. However, we believe that this again led to the loss of important contextual information. To fix this,

[6] https://www.scipy.org/

we independently extract character n-gram features from the title and description. This step yields a title feature vector and a description feature vector. We then horizontally concatenate these vectors, forming a single feature vector which includes separate feature sets for the title and description.

Results

We consider 10 marketplaces to train and test our learning model. A summary of these marketplaces is shown in Table 5.3. Table 5.4 gives instance of products defined as being relevant or not. With the help of security experts we label 25% of the products from each marketplace. The experimental setup is as follows. We perform a leave-one-marketplace-out cross-validation. In other words, given n marketplaces we train on $n - 1$ and test on the remaining one. We repeat this experiment for all the marketplaces. For the supervised experiment we only use the 25% labeled data from each marketplace. We evaluate the performance based primarily on three metrics: precision, recall and unbiased F1. Precision indicates the fraction of products that were relevant from the predicted ones. Recall is the fraction of relevant products retrieved. The results are averaged and weighted by the number of samples in each market. In this application a high recall is desirable as we do not want to omit relevant products. In the supervised approaches SVM with linear kernel performed the best, recalling 87% of the relevant products while maintaining a precision of 85% (Fig. 5.4). SVM performed the best likely due to the fact it maximizes generality as opposed to minimizing error.

As stated, only 25% of the data is labeled, as labeling often requires expert knowledge. However, this significant cost and time investment can be reduced by applying a semi-supervised approach which leverages the unlabeled data to aid in classification. It takes approximately one minute for a human to label 5 marketplace products or 2 topics on forums as relevant or not, highlighting the costliness of manual labeling. The experimental setup

Table 5.3: Markets and Number of products collected.

Markets	Products
Market-1	439
Market-2	1329
Market-3	455
Market-4	4018
Market-5	876

Markets	Products
Market-6	497
Market-7	491
Market-8	764
Market-9	2014
Market-10	600

Table 5.4: Example of Products.

Product Title	Relevant
20+ Hacking Tools (Botnets Keyloggers Worms and More!)	YES
SQLI DUMPER V 7.0 SQL INJECTION SCANNER	YES
Amazon Receipt Generator	NO
5 gm Colombian Cocaine	NO

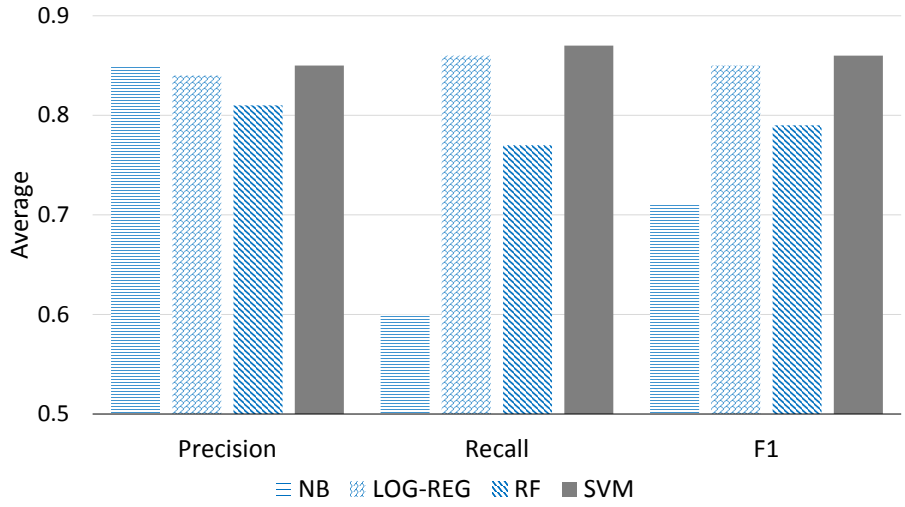


Fig. 5.4: Average Precision, Recall and F1 comparisons for NB, LOG-REG, RF and SVM for product classification.

is similar to the supervised approach, but this time we also utilize the large unlabeled data from each marketplace (75%) for training.

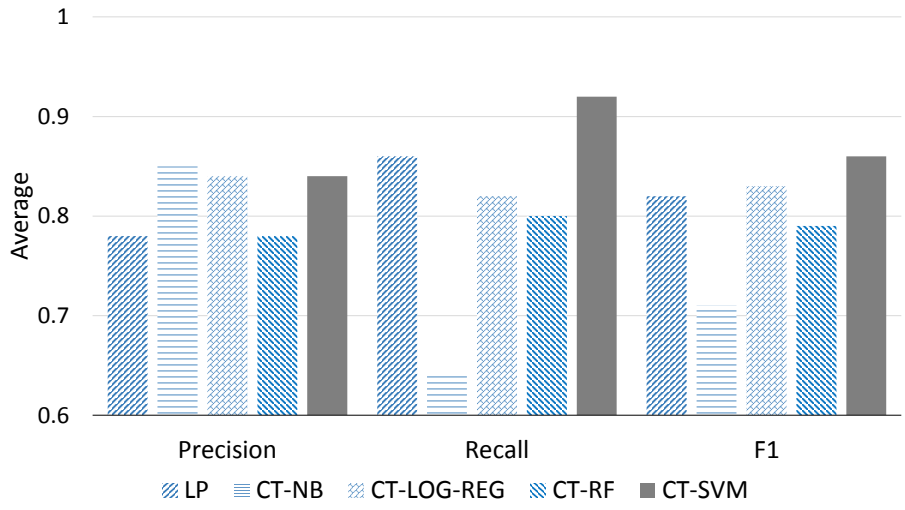


Fig. 5.5: Average Precision, Recall and F1 comparisons for LP, CT-NB, CT-LOG-REG, CT-RF and CT-SVM for product classification.

Fig. 5.5 shows the performance comparison for the semi-supervised approaches. For the co-training approach we divide the feature space into two sets. The two feature sets used are both based on character n-grams. However, the set of words from which the character n-grams are derived are disjoint between the two sets. In this way, the two corresponding feature vectors can be treated as being independent from one another. Hence we get two views of the same sample. Co-training with Linear SVM is able to recall 92% of the relevant products as compared to label propagation and other variants of co-training while maintaining a precision of 82%, which is desirable. In this case, the unlabeled data aided the classification in improving the recall to 92% without significantly reducing the precision.

5.4.3 Experiment: Forums

In addition to the darkweb/deepweb marketplaces that we have already discussed, there are also numerous darkweb forums on which users will not only discuss on malicious hacking related topics. Again, there is the issue that only a fraction of these topics with posts on these forums contain information that is relevant to malicious hacking or the trading of exploits. hence, we need a classifier to identify relevant topics. This classification problem is very similar to the product classification problem previously discussed, with similar set of challenges.

We performed evaluation on an English forum and a Russian forum. For the English forum we considered a dataset of 781 topics with 5373 posts. Table 5.5 gives instance of topics defined as being relevant or not. We label 25% of the topics and perform a 10-fold cross validation using supervised methods. We show the results from the top two performing supervised and semi-supervised methods. In the supervised setting, LOG-REG performed the best with 80% precision and 68% recall (Fig. 5.6). On the other hand, leveraging unlabeled data in a semi-supervised technique improved the recall while maintaining the precision. We note that in this case the 10-fold cross validation was performed only on the labeled points. In the semi-supervised domain co-training with LOG-REG improved the recall to 80% with precision of 78%.

Table 5.5: Example of Topics.

Topic	**Relevant**
Bitcoin Mixing services	YES
Hacking service	YES
I can vend cannabis where should I go?	NO
Looking for MDE/MDEA shipped to Aus	NO

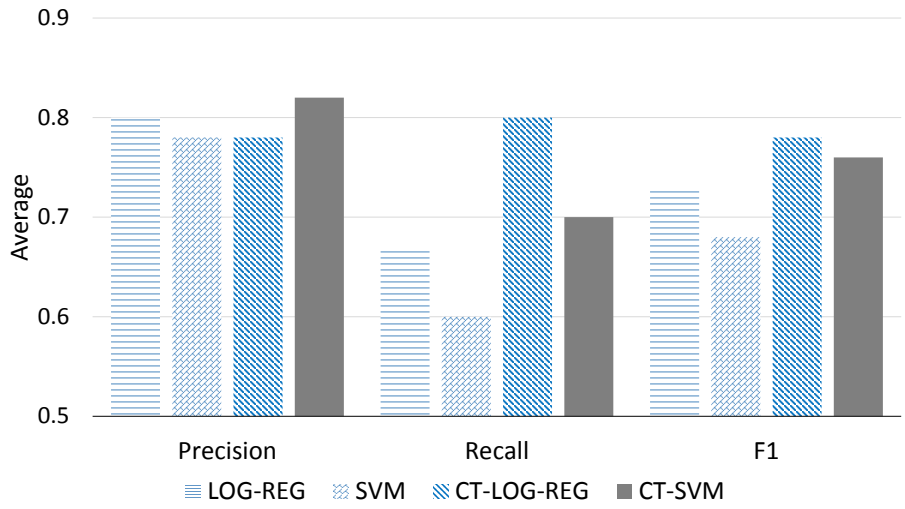


Fig. 5.6: Average Precision, Recall and F1 comparisons for LOG-REG, SVM, CT-LOG-REG, and CT-SVM for English forum topic classification.

We also encounter forums in languages other than English. Many of the non-English forums like Russian use English words to describe hacking techniques and exploits (e.g. "RAT", and "botnet"). Hence, we use the same character n-gram features for the Russian forum too. For evaluation we consider a Russian forum with 1609 topics comprising of 8961 posts. We had 25% of the topics labeled by a Russian speaking security analyst. The experimental setup is similar to English forums. The comparison is shown in Fig. 5.7. We note that supervised methods do better than semi-supervised methods in this setting. LOG-REG has the best recall of 58% with 60% precision. We are exploring a combination of machine learning and keyword filtering to improve the performance on foreign language forums.

5.4.4 Experiment: Subreddits

We also crawl data from subreddits where discussion is focused on darkweb and deepweb websites. But, just as in the case of marketplaces/forum, not all subreddit posts are relevant

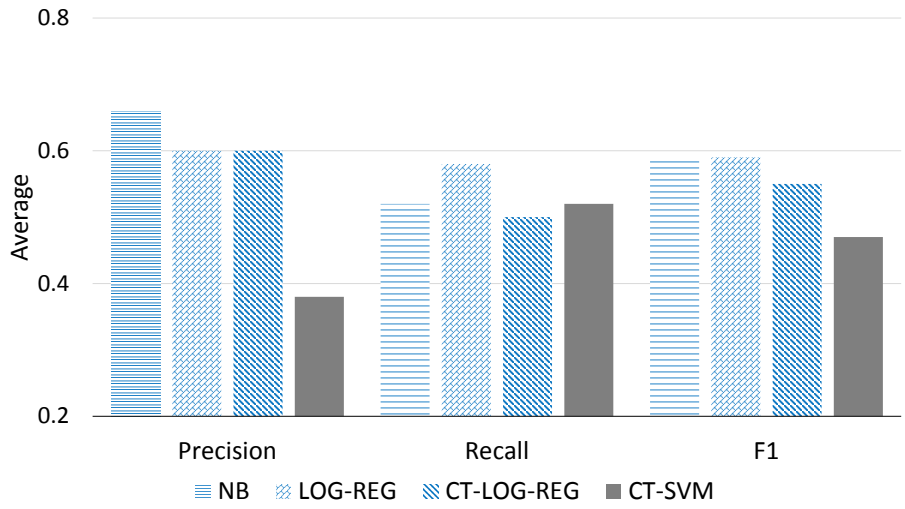


Fig. 5.7: Average Precision, Recall and F1 comparisons for NB, LOG-REG, CT-LOG-REG and CT-SVM for Russian forum topic classification.

to malicious hacking. Thus, we need a classifier to identify the relevant ones. For evaluating the model we consider 1550 topics with around 8000 posts from 33 subreddits. We consider the topic to be relevant if atleast one of the posts in the topic is relevant. We label 25% of the topics. Using these labeled samples we perform a 10-fold cross validation and average the results. Fig. 5.8 shows the two best performing supervised methods (NB and LOG-REG). Naive Bayes is able to recall 68% of the relevant subreddits with a precision of 53%.

On the other hand, leveraging unlabeled data in a semi-supervised technique improves precision while maintaining the recall. Fig. 5.8 shows the two best performing semi-supervised methods (CT-LOG-REG and CT-SVM) in comparison with the supervised methods. Here, the 10-fold cross validation is performed only on the labeled points. Co-training with linear SVM performs the best with an average precision of 74%, recalling 68% of the subreddits. Hence, again it provides a significant performance increase in precision. As with the marketplace classifier, we used two feature vectors of character n-grams derived from two disjoint sets of words.

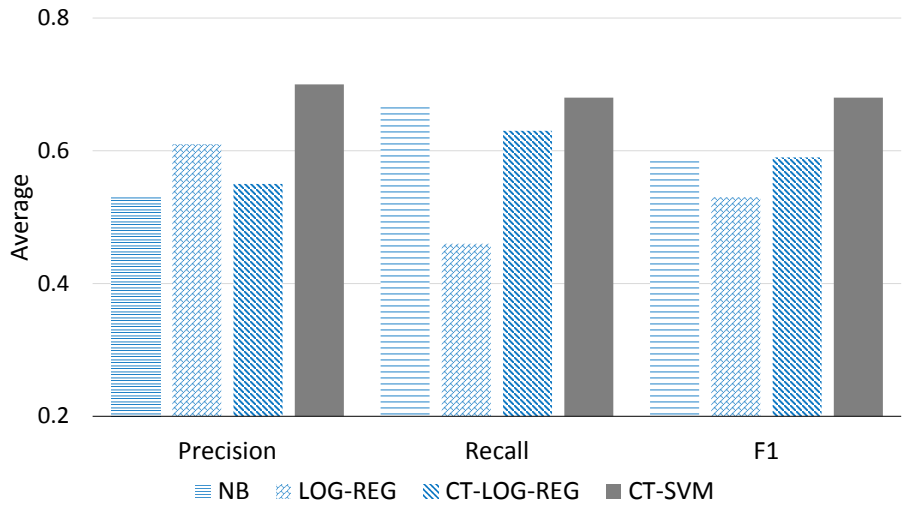


Fig. 5.8: Average Precision, Recall and F1 comparisons for LP, CT-LOG-REG, CT-RF and CT-SVM for subreddits.

5.4.5 Darknet New Page Discovery

The crawler often encounters new web links while crawling data from forums and subreddits. These new links might point to new marketplaces or forums relating to malicious hacking. We want to automate the process of determining whether or not a given new web-page is relevant to our data collection efforts. For the previous classification problems we were able to rely on the structure of the input. That is, in the case of the product classifier we knew that we were receiving only a product title and description. In the case of the forum/subreddit classifier we had a thread title and the post content. This classification problem is different. The input is a single HTML page, the structure of which is completely unknown. The classifier then has to determine if the page is relevant or not. Without any assumption on structure, it is very difficult to extract only the parts of the page that are relevant. In the pre-processing step we extract all visible text on the page (including header, footer, sidebar, menu etc.).

In our first approach, we used a bag-of-words approach with TF-IDF (term frequency

- inverse document frequency) on the extracted text (i.e. all visible text on the webpage). Using this approach, when given two pages from the same web site, both pages will have an identical header, footer, sidebar, menu, etc. One difficulty here was that the term frequencies of the words in the header, footer, etc. are not necessarily that important for classification. This generates a lot of noise in the feature space. With TF-IDF-based features, we found that the classifiers greatly overfit to the pages that were in the positive training set. As a result, pages from new sites were nearly always classified as negative – regardless of content. To help mitigate the problem of overfitting models to sites in the training set, we used the topic-distribution generated by Latent Dirichlet Allocation (LDA) [12] as features, rather than TF-IDF or bag-of-words, which greatly improved performance on an independent evaluation set. In our ongoing work, we are examining additional features beyond those based on text.

Results

For training our model, we use all the positive webpages from marketplaces and forums already identified as relevant by an analyst. The negative pages were gathered by an analyst during their search for relevant darkweb and deepweb websites. Using all of the labeled pages as a training set we trained two Linear SVM models, one with TF-IDF features and one with LDA topic distribution features, with word stemming done for both the models. To evaluate the models, a security analyst provided us with a list of links. The links were crawled, yielding 2855 HTML pages with unknown content. Hence, the evaluation described in this section was a true validation set. Our focus is on precision with the intuition that the classifier can point out a relatively small number of pages that are likely to be relevant to the analyst.

TF-IDF-Based Results. When using standard TF-IDF as features, only 35 pages were classified as relevant and, of them, only 3 unique sites were represented, two of which were

deemed irrelevant by an analyst and the third site having pages that appeared in the training set. Using TF-IDF as features yielded no pages for new markets.

LDA-Based Results. LDA greatly improved performance on the evaluation set. When evaluating a Linear SVM model with LDA topic distribution features, trained on the labeled data, 58 of the 2855 unlabeled pages were given a positive classification. Of the 58 pages, the analyst deemed 50 of them as relevant to what they typically look for, with seven new markets represented in the set of 58 pages. This classifier performed far better at extracting the "market structure," as nearly all positively classified pages were from darkweb markets. Once markets have been identified, we can leverage the market product classifier discussed previously to only extract products relating to malicious hacking. We also note that the LDA topics themselves are useful to the analyst - especially as topics evolve over time. Table 5.6 shows a sample of the 25 LDA topics that were used in this experiment.

Table 5.6: A sample of Positive Topics.

1. bitcoin, use, address, account, order, contact, email, service, product, please, day, send, market, new, make, share, time, free, month
2. price, day, ago, item, usd, btc, fix, quantity, buy, view, left, bid, unlimited, software, book, ms, fraud, secure, exploit
3. service, onion, hidden, tor, bitcoin, forum, wiki, host, link, card, mark0et, directory, clearnet, site, drug, web, marketplace

5.5 Case Studies

We analyze the data with the purpose of answering the questions raised in the Section 5.1. We will be using the following key security terms. *Vulnerability* is a security flaw that allows an attacker to compromise a software or an operating system. *Exploit* is a piece of software that takes advantage of a vulnerability in a piece of software or operating system

to compromise it. *Patch* is a piece of software used to improve existing software by fixing vulnerabilities to improve security. We discuss the following case-studies.

5.5.1 Discovery of Zero-Day Exploits.

Over a 4 week period, we detected 16 zero-day exploits from the marketplace data. Zero-day exploits leverage vulnerabilities that are unknown to the vendor. Table 5.7 shows a sample of zero-day exploits with their selling price in Bitcoin. The Android WebView zero-day affects a vulnerability in the rendering of web pages in Android devices. It affects devices running on Android 4.3 Jelly Bean or earlier versions of the operating system. This comprised of more than 60% of the Android devices in 2015. After the original posting of this zero-day, a patch was released in Android KitKit 4.4 and Lollipop 5.0 which required devices to upgrade their operating system. As not all users have/will update to the new operating system, the exploit is continues to be sold for a high price. Detection of these zero-day exploits at an earlier stage can help organizations avoid an attack on their system or minimize the damage. For instance, in this case, an organization may decide to prioritize patching, updating , or replacing certain systems using the Android operating system.

Table 5.7: Example of Zero-day exploits.

Zero-day exploit	**Price (BTC [7])**
Internet Explorer 11 Remote Code Execution 0day	20.4676
Android WebView 0day RCE	40.8956
Fresh 0day MS Office	38.3436

5.5.2 Exploits targeting known vulnerabilities.

Zero-day vulnerabilities are difficult to discover, hence the zero-day exploits are rare. But exploits targeting known vulnerabilities often show up on the marketplaces for sale.

These exploits are advertised to target specific vulnerabilities. Sometimes vendors mention Common Vulnerability and Exposure (CVE) numbers assigned by the National Institute of Standards and Technology (NIST). Using NIST's National Vulnerability Database (NVD) [8] , we can determine the vulnerability and the target softwares from the CVE number. For instance, the Silent Doc exploit allows remote attackers to execute arbitrary code or cause a denial of service (memory corruption). It affects Microsoft Word. The severity level was listed HIGH on this exploit. Exploit kit on the other hand targets many vulnerabilities and is expensive. The Xer Exploit Kit (Table 5.8) targets 7 vulnerabilities. Also note that Microsoft also assigns vulnerability numbers for its products with a "Microsoft Security Bulletin" (MSB) number. These numbers are sometimes seen as well in marketplace product descriptions.

Table 5.8: Exploit-Vulnerability.

Exploit	**Vulnerability**
SILENT DOC EXPLOIT	CVE-2014-1761
Sqlninja	CVE-2010-0232
Xer Exploit Kit / traffic / LOADS	CVE-2015-2426, CVE-2015-0313, CVE-2015-0311, CVE-2014-0556, CVE-2015-0317, CVE-2014-0515, CVE-2015-2444

5.5.3 Users having presence in markets/ forums.

Previous studies on darkweb crawling [41, 10] explore a single domain, namely forums. We create a social network that includes both types of information studied in this chapter: marketplaces and forums. We can thus study and find these cross-site connections that were previously unstudied. We are able to produce this connected graph using the "usernames"

[8] https://nvd.nist.gov/home.cfm

used by vendors and users in each domain. A subgraph of this network containing some of the individuals who are simultaneously selling products related to malicious hacking and publishing in hacking related forums is shown in Fig. 5.9. In most cases the vendors are trying to advertise/discuss their products on the forums, demonstrating their expertise. Using these integrated graphic representations, one can visualize the individuals' participation in both domains, making the right associations that lead to a better comprehension of the malicious hacker networks. It is helpful in determining social groups within the forums of user interaction. The presence of users on multiple markets and forums follows a power law. From Fig. 5.10, majority of users only belong to a single market or forum. We note that there are 751 users that are present in more than two platforms. Fig. 5.11 considers one such user/vendor. The vendor is active in 7 marketplaces and 1 forum . The vendor offers 82 malicious hacking related products and discusses these products on the forum. The vendor has an average rating of 4.7/5.0, rated by customers on the marketplace with more than 7000 successful transactions, indicating the reliability of the products and the popularity of the vendor.

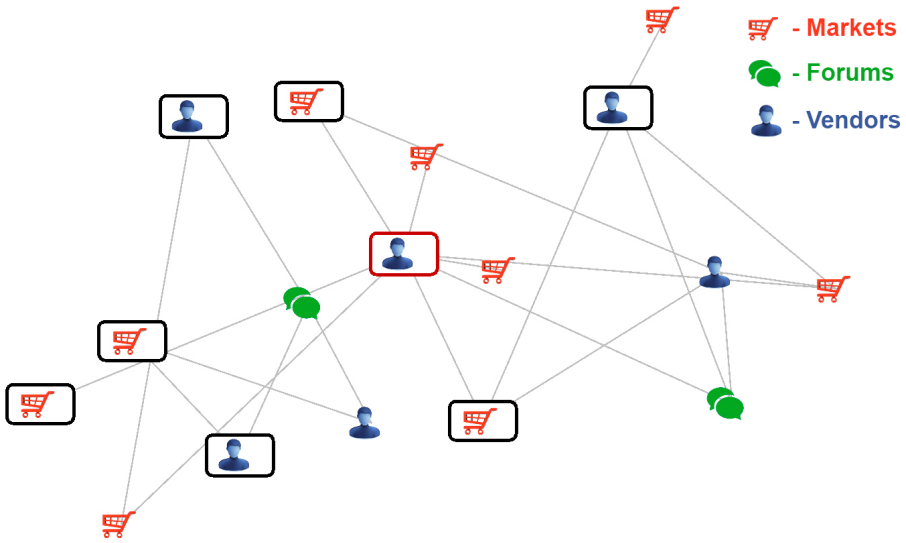


Fig. 5.9: Vendor/User network in marketplace and forum.

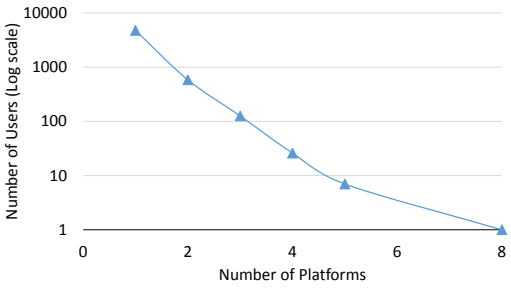


Fig. 5.10: Users in multiple markets and forums.

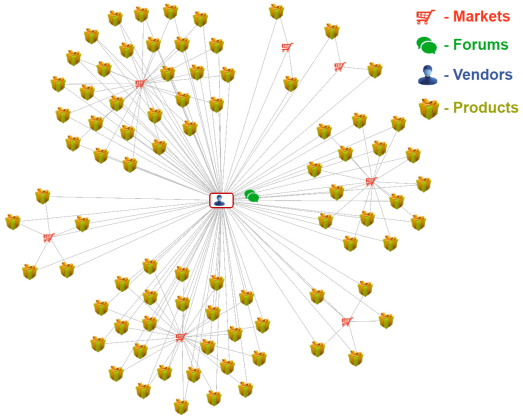


Fig. 5.11: A centric network of a Vendor.

5.6 Related Work

Web crawling is a popular way of collecting large amounts of data from the Internet. In many applications researchers are interested in specific topics for their application. Hence the need for a topic-based crawler popularly referred to as a focused crawler [24, 23]. Most of the focused crawlers are designed to collect information from the *surface web* with little concentration on the darkweb websites. More recently a focused crawler concentrating on dark web forums was designed [41]. This research primarily concentrated on forums, collecting data over a period of time and then performing static analysis to study online communities. The authors also describe different data mining techniques for these forums in [26]. We, on the other hand, not only look at darkweb forums but also collect information from marketplaces hosting a range of products relating to malicious hacking.

Additionally, web-crawlers have been developed to aid law enforcement to track online extremist activities [82]. This work has included the use of a self-guided web-crawler using sentiment analysis to identify extremist content, threats to critical infrastructure [75] and online sexual child exploitation [135]. Another application of leveraging darkweb information to counter human trafficking is developed by DARPA through the Memex

program [9] - a program with different goals than the work described in this chapter.

Previous work leverages the exploit information from marketplaces in a game theoretic framework to formulate system configurations that minimize the potential damage of a malicious cyber attack [104]. Other work suggests that hacker communities can be analyzed to aid in detection to reveal existing and emerging threats. Threats that pose great risk to individuals, businesses, and government [10]. It further states that knowledge is distributed in forums. That minimally skilled people could learn enough by simply frequenting such platforms. This behavior is widespread geopolitically (namely across US, China, Russia, the Middle-East). Studying these hacker communities gives insights in the social relationships. Also, the distribution of information amongst users in these communities based on their skill level and reputation [52, 57, 50]. These forums also serve as markets where malware and stolen personal information are shared/ sold [51]. Samtani et al. analyze hacker assets in underground forums [108]. They discuss the dynamics and nature of sharing of tutorials, source code, and "attachments" (e.g. e-books, system security tools, hardware/software). Tutorials appear to be the most common way of sharing resources for malicious attacks. Source code found on these particular forums was not related to specific attacks. Mostly general, SQL-related (suggesting targets like databases of banks). Additionally underground forums have also been analyzed to captures the dynamic trust relationships forged between mutually distrustful parties [84]. They analyze six different underground clearnet forums - examining the properties of the social networks formed within. The content of the goods and services being exchanged. Lastly, how individuals gain and lose trust in these underground forums. These applications gather information from the clearnet to test their theories. Less effort is put towards analyzing darkweb information given the challenges in gathering information from the darkweb. Additionally, our focus in this work is on the unique characteristics of forums and markets supporting malicious hacking in particular - not

[9] https://opencatalog.darpa.mil/MEMEX.html

general illicit activities.

5.7 Summary

In this chapter we implement a system for intelligence gathering related to malicious hacking. We consider social platforms on darkweb and deepweb for data collection. We address various design challenges to develop a focused crawler using data mining and machine learning techniques. We transitioned this system to a commercial partner to increase the scale of data collection and maintain the system. The constructed database is made available to security professionals in order to identify emerging cyber-threats and capabilities as demonstrated by the application in Chapter 6.

Chapter 6

AT-RISK SYSTEM IDENTIFICATION VIA ANALYSIS OF DISCUSSIONS ON THE DARKWEB

6.1 Introduction

Adequate assessment of threats to systems is a central aspect of a mature security policy—identifying systems that are at-risk can help defend against potential cyber attacks. Currently, organizations rely on the rating system (CVSS score) provided by The National Institute of Science and Technology that maintains a comprehensive list of publicly disclosed vulnerabilities in the National Vulnerability Database (NVD [86]) to identify if their systems are at risk. Case studies have shown poor correlation between the CVSS score and the likelihood that a vulnerability on a system will be targeted by hackers [2]. Hence, organizations are constantly looking for ways to proactively identify if their vulnerable systems are of interest to hackers.

Table 6.1: System components and examples

Components	**Explanation and Examples**
Platform	Can be either hardware (h), operating system (o), or application (a) based on what the vulnerability exploits.
Vendor	The owner of the vulnerable product. Examples include Google, Microsoft, The Mozilla Foundation, and the University of Oxford.
Product	The product that is vulnerable. Examples include Internet Explorer, Java Runtime Environment, Adobe Reader, and Windows 2000.

Threat intelligence from darkweb (D2web) has been leveraged to predict whether or

not a vulnerability mention on D2web will be exploited [4, 3]. This method only considers hacker discussions that have a CVE number mentioned in them—a limitation of the approach is therefore that discussions with no vulnerability identifiers (CVE) that are of interest to threat actors are not taken into account. In this chapter, we propose to leverage this threat intelligence gathered from D2web markets and forums to identify the systems that might be of interest to threat actors. We identify systems based on the structured naming scheme Common Platform Enumeration (CPE [31]). We focus our efforts towards identifying the first three system components of the CPE naming scheme; Table 6.1 shows these three components, with examples for each.

We design a system that leverages threat intelligence (hacker discussions) and makes a decision regarding at-risk systems, at the same time providing arguments as to *why* a particular decision was made. It explores multiple competing hypotheses (in this case multiple platforms, vendors, products) based on the discussions for and against a particular at-risk component. The resulting system is a hybrid that combines DeLP with machine learning classifiers. Previously, a similar reasoning system was employed for attributing cyber-attacks to responsible threat actors in chapter 4 evaluated on a capture-the-flag dataset in chapter 3.

In this chapter:

- We frame identifying at-risk systems as a multi-label classification problem, and apply several machine learning approaches to compare their performance. We find that large number of possible label choices for vendors and products with less representation in training account for the majority of the misclassified samples.

- To address misclassification, we propose a hybrid reasoning framework that combines machine learning techniques with defeasible argumentation to reduce the set of possible labels for each system component. The reasoning framework can provide

arguments supporting the decisions, indicating *why* a particular system was identified over others; this is an important aspect, supporting a security analyst in better understanding the result.

- We report on experiments showing that the reduced set of labels used in conjunction with the classifiers leads to significant improvement in precision (15%-57%) while maintaining comparable recall.

6.1.1 Vulnerability related terms

Vulnerability is a flaw in a system (software/hardware) that makes the system vulnerable to attacks compromising the confidentiality, integrity or availability of the system to cause harm [99].

CVE: Common vulnerability enumeration (CVE) is a unique identifier assigned to a system vulnerability reported to NIST [32]. NIST maintains a database of all the vulnerabilities publicly available in the National Vulnerability Database (NVD [86]). Predicting exploitability of a CVE is an important problem and recent work leveraging darkweb data has shown good performance in achieving that goal [4, 3]. But these techniques reply on direct mentions of CVE's. We Note that a very small portion of hacker discussions in the data from the commercial provider has direct CVE mentions.

CPE: Common platform enumeration (CPE) is a list of software / hardware products that are vulnerable for a given CVE. NIST makes this data available for each vulnerability in its database. Identifying at-risk systems in terms of its components (see Table 6.1) is an important step towards predicting if those systems will be targeted by threat actors (in cases where the hacker discussion is not associated with a CVE number). For the system components under consideration, there exists a hierarchy starting from the platform to vendor to product. For instance, if we are considering operating systems, then there are

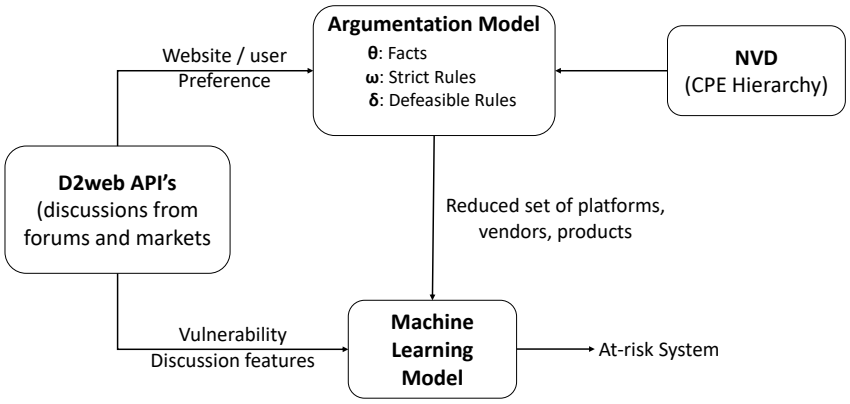


Fig. 6.1: Reasoning System

limited number of vendors that provide it: Microsoft, Apple, Google, etc. If we identify Microsoft as our vendor, then the products are related to the Windows operating system. This hierarchy helps us to narrow down possible choices as we go down the hierarchy.

6.2 System Overview

Fig. 6.1 gives an overview of the reasoning system; it consists of the following three main modules:

- **Knowledge Base:** Our knowledge base consists of hacker discussions from darkweb (D2web) forums and marketplaces collected in chapter 5. This data is maintained and made available through APIs by a commercial darkweb threat intelligence provider [1] . The database is collected from 302 websites [2] . We use the hacker discussions in terms of posted content (from forums) and item descriptions (from markets), the website it is posted on, and the user posting the discussion as inputs to both the

[1]Cyber Reconnaissance, Inc. (CRY3CON), https://www.cyr3con.com.

[2]At the time of writing

argumentation and machine learning models. We also input the CPE hierarchy from NVD to the argumentation model. We discuss and provide further analysis of the data in Section 2.2. For the experiment, we sort the dataset by time (depending on when the discussion was posted); the first 80% is reserved for training (knowledge base) and the remaining 20% for testing. We follow similar time split to compute the CPE hierarchy as well.

- **Argumentation Model:** This component constructs arguments for a given query (at-risk system component) using elements in the knowledge base. We use a formalism called DeLP that combines logic programming with defeasible argumentation. It is made up of three constructs: *facts:* observations from the knowledge base that cannot be contradicted; *strict rules:* logical combinations of facts that are always true; and *defeasible rules:* can be thought of as strict rules but are only true if no contradictory evidence is present. We discuss the argumentation framework with examples for each of the constructs in Section 3.3. Arguments help reduce the set of possible choices for platforms, vendors and products; this reduced set of possible system components acts as one of the inputs to the machine learning model. The argumentation model thus constrains the machine learning model to identify the system from the reduced set of possible platforms, vendors, and products.

- **Machine Learning Model:** The machine learning model takes the knowledge base and query as input, along with the reduced set of possible system components from the argumentation model, and provides a result identifying the system. It is constrained by the argumentation model to select the components from the reduced platform, vendor and product set, which aids the machine learning model (improving precision) as demonstrated in the results section of the chapter. We use text-based features extracted from the discussions (TF-IDF/Doc2Vec) for the machine learning model.

Any standard machine learning model can be used in this module. We provide a comparison of different machine learning models to select the best one.

6.3 Dataset

6.3.1 D2web data

We use D2web data supplied by a threat intelligence company collected in chapter 5. The data is accessed via APIs. The data is comprised of forum discussions and marketplace items offered for sale in D2web. Exploration of D2web discussions in terms of their structure, content and behavior of users who post these discussions is reported in [112]. The data is collected periodically to obtain time-based information indicating changes in the forums and marketplaces. To ensure collection of cyber-security relevant data, machine learning models are employed that filter the data related to drugs, weapons, and other irrelevant discussions. Table 6.2 shows the characteristics for the websites, posts/items, and users. The data is comprised from websites with different languages. A single website might have discussions in different languages. Fig. 6.2 shows the percentage of total websites from the D2web for the top ten languages used to post discussions Majority of the websites have discussions in English (73%), with other languages having an even distribution. The commercial data collection platform automatically identifies the language and translates it to English using the Google Translate API [46].

Ground Truth. In order to evaluate the performance of the reasoning framework, we need ground truth associated with the hacker discussions. To obtain ground truth we consider discussions from forums and marketplaces that mention a CVE number. From the CVE number we can look up the vulnerable systems using the NVD; we note that for both training and testing we remove the CVE number while computing features. Table 6.2 shows the characteristics for the websites, posts/items, and users that mention a CVE number. The

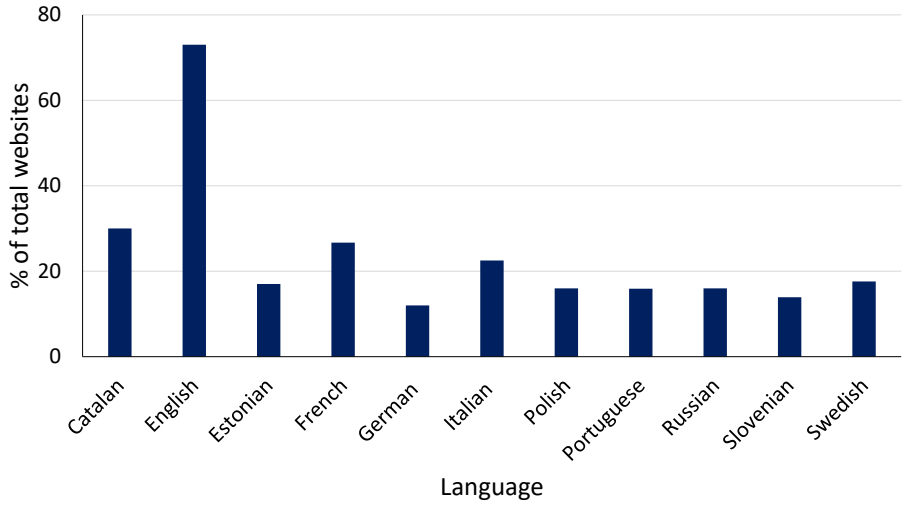


Fig. 6.2: Percentage of total websites belonging to the top ten languages in the D2web data.

Table 6.2: Characteristics of D2web data

Number of D2web websites	302
Number of unique users	635,163
Number of unique posts / items	6,277,638
Number of D2web websites (CVE mentions)	135
Number of unique users (CVE mentions)	3,361
Number of unique posts / items (CVE mentions)	25,145

hacker discussion with CVE mentions belong to 135 websites posted by 3361 users. On analyzing the CVE mentions most of the older vulnerabilities target products that are no longer in use. For that reason in our experiments we consider CVE discussions posted after 2013 (starting 01/01/2014). These discussion make up around 70% of the total CVE discussions.

CPE Hierarchy. We compute the hierarchy for all the vulnerable systems from all the vulnerabilities disclosed in NVD [86], and maintain it as a dictionary to build arguments on top of it. Fig. 6.3 shows a subset of the built hierarchy with the three system components

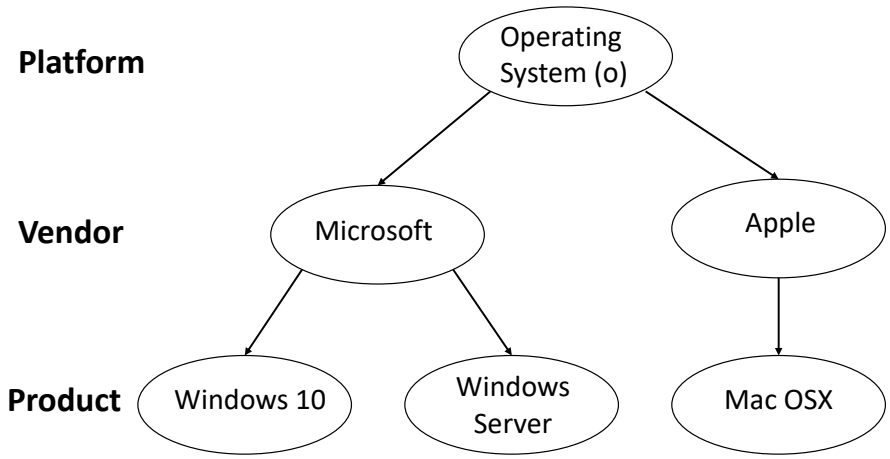


Fig. 6.3: Subset of CPE Hierarchy

(platform, vendor and product).

Website/User preference. We compute and maintain a list of system components discussed for each website and user. This lets us know if a particular website is preferred by hackers to discuss specific at-risk systems. The user list gives us the preference of the user regarding what at-risk systems are of interest to him/her.

Overall in our dataset, for platforms most discussions pose a threat to operating systems (57%), following by applications (43%) and hardware makes up a small fraction of the discussions (3%). There are discussions that pose a risk to multiple platforms i.e. operating systems and application or in few instances all three. For vendors, the top five at-risk based on CVE mentions in the hacker discussions: Microsoft (24%), Linux (9%), Apple (6%), Oracle (5%), Adobe (5%). Similar to platforms discussions can pose a risk to multiple vendors. For products the distribution is more even since a single vendor can have multiple products. Even though Microsoft dominates the vendor discussion, it also has the most number of products that are at risk. The top five at-risk products based on CVE mentions

in the hacker discussions: Windows server (5%), Windows 8.1 (4%), Linux kernel (3.8%), Mac OSX (2.3%), Flash player (1.9%).

6.4 Argumentation Model

Our approach relies on a model of the world where we can analyze competing hypotheses. Such a model allows for contradictory information so it can handle inconsistency in the data similar to the one employed for attributing cyber-attacks to responsible threat actors [115, 94].

Before describing the argumentation model in detail, we introduce some necessary notation (similar to be one used in chapter 3). Variables and constant symbols represent items such as the platform/vendor/product at-risk by the discussion and post/webID/userID represent the hacker discussion, where it was posted and who posted it respectively (we note that for privacy concerns the webID/userID is represented as an integer in the data provided by the APIs—the names are not disclosed). We denote the set of all variable symbols with $\mathbf{V}$ and the set of all constants with $\mathbf{C}$. For our model we require six subsets of $\mathbf{C}$:

- $\mathbf{C}_{post}$ denoting the hacker discussion,
- $\mathbf{C}_{web}$, denoting the websites (both forums and marketplaces) where the hacker discussion was posted,
- $\mathbf{C}_{user}$, denoting the users who posts hacker discussions, and
- $\mathbf{C}_{platform}$, $\mathbf{C}_{vendor}$, $\mathbf{C}_{product}$ denoting the three components at-risk by the discussion (see Table 6.1).

We use symbols in all capital letters to denote variables. In the running example, we use a subset of the D2web dataset collected by the threat intelligence company.

Table 6.3: Example predicates and explanation

Predicate	**Explanation**
posted($post_1$, $webID_1$)	$post_1$ was posted on the website $webID_1$.
at_risk($\mathcal{D}$, Y)	Post $\mathcal{D}$ discussed vendor Y being at-risk.
user_preference ($userID_1$, $microsoft$)	$userID_1$ prefers to post discussions regarding Microsoft systems at-risk.
previously_seen ($webID_1$, $adobe\_flash$)	At-risk discussions regarding Adobe Flash are discussed in $webID_1$.
parent($microsoft$, $safari$)	Vendor Microsoft is a parent of product Safari.

Example 5. *The following system and post/web/user information will be used in the running example:*

$\boldsymbol{C}_{post} = \{post_1, post_2, ..., post_n\}$

$\boldsymbol{C}_{web} = \{webID_1, webID_2, ..., webID_n\}$

$\boldsymbol{C}_{user} = \{userID_1, userID_2, ..., userID_n\}$

$\boldsymbol{C}_{platform} = \{h, o, a\}$

$\boldsymbol{C}_{vendor} = \{microsoft, google, the\_mozilla\_foundation\}$

$\boldsymbol{C}_{product} = \{internet\_explorer, windows\_10, adobe\_reader\}$

The language also contains a set of predicate symbols that have constants or variables as arguments, and denote events that can be either *true* or *false*. We denote the set of predicates with $\mathbf{P}$; examples of predicates are shown in Table 6.3. For instance, $user\_preference(userID_1, microsoft)$ will either be true or false, and denotes the event where $userID_1$ prefers to post discussions regarding *microsoft* systems at-risk.

A *ground atom* is composed by a predicate symbol and a tuple of constants, one for each argument—hence, ground atoms have no variables. The set of all ground atoms is denoted

with $\mathbf{G}$. A *ground literal* L is either a ground atom or a negated ground atom. An example of a ground atom for our running example is $posted(\textit{post}_1, \textit{webID}_1)$. In the following, we will use $\mathbf{G}'$ to denote a subset of $\mathbf{G}$.

Defeasible Logic Programming: DeLP is a formalism that combines logic programming with defeasible argumentation; we refer the interested reader to [42] for a fully detailed presentation of the system. The formalism is summarized in chapter 3 (see Section 3.3) along with the three constructs, namely *facts*, *strict rules*, and *defeasible rules*. These three constructs are used to build *arguments*, and DeLP programs are simply sets of facts, strict rules and defeasible rules. We adopt the usual notation for DeLP programs, denoting the program (or knowledge base) with $\Pi = (\Theta, \Omega, \Delta)$, where Θ is the set of facts, Ω is the set of strict rules, and Δ is the set of defeasible rules. Examples of the three constructs are provided with respect to the dataset in Fig. 6.4. We now describe the notation used to denote these constructs. We reiterate the definitions of the three constructs.

Facts (Θ) are ground literals that represent atomic information or its (strong) negation ($\neg$).

Strict Rules (Ω) represent cause and effect information; they are of the form $L_0 \leftarrow L_1, ...L_n$, where L_0 is a literal and $\{L_i\}_{i>0}$ is a set of literals.

Defeasible Rules (Δ) are weaker versions of strict rules, and are of the form $L_0 \prec L_1, ..., L_n$, where L_0, is the literal and $\{L_i\}_{i>0}$ is a set of literals.

When a hacker discussion happens on D2web, the model can be used to derive arguments to determine the at-risk system (in terms of platform, vendor, and product). Derivation follows the same mechanism as classical logic programming [73]; the main difference is that DeLP incorporates defeasible argumentation, which decides which arguments are warranted, which arguments are defeated, and which arguments should be considered to be *blocked*—the latter are arguments that are involved in a conflict for which a winner cannot be determined.

Fig. 6.4 shows a ground argumentation framework demonstrating constructs derived from our D2web data. For instance, θ_1 indicates the fact that a hacker discussion $post_1$ was posted on the D2web website $webID_1$, and θ_5 indicates that user $userID_1$ prefers to post discussions regarding $apple$ products. For the strict rules, ω_1 says that for a given post $post_1$ posing a threat to operating system (o), the vendor *sandisk* cannot be at risk if the parent of *sandisk* is not operating system (o) [3] . Defeasible rules can be read similarly; δ_2 indicates that if $post_1$ poses a threat to the vendor *apple*, the product *safari* can be at-risk if *apple* is the parent of *safari*. By replacing the constants with variables in the predicates we can derive a non-ground argumentation framework that can be applied in general.

The following examples discuss arguments for our scenario.

Example 6. *Fig. 6.5 shows example arguments based on the KB from Fig. 6.4; here,* $\langle \mathcal{A}_3,$ $\mathsf{at\_risk}(post_1, apple)\rangle$ *is a subargument of* $\langle \mathcal{A}_2,$ $\mathsf{at\_risk}(post_1, safari)\rangle$.

We engineer our at-risk system framework as a set of defeasible and strict rules whose structure was created manually, but are dependent on values learned from a historical corpus of D2web data. Then, for a given post discussing a vulnerability, we instantiate a set of facts for that situation; this information is then provided as input into the DeLP system, which uses heuristics to generate all arguments for and against every possible components of the system (platforms, vendors, products) for the post discussion. Dialectical trees based on these arguments are analyzed, and a decision is made regarding which components are *warranted*. This results in a *reduced set* of potential choices, which we then use as input into a classifier to obtain the at-risk system. The following section discusses these steps in full detail.

[3] This encodes the CPE hierarchical structure.

Θ :

$$\theta_1 = \mathsf{posted}(post_1, webID_1)$$

$$\theta_2 = \mathsf{posted}(post_1, userID_1)$$

$$\theta_3 = \mathsf{parent}(o, microsoft)$$

$$\theta_4 = \mathsf{parent}(apple, safari)$$

$$\theta_5 = \mathsf{user\_preference}(userID_1, apple)$$

$$\theta_6 = \mathsf{previously\_seen}(webID_1, o)$$

Ω :

$$\omega_1 = \neg\, \mathsf{at\_risk}(post_1, sandisk) \leftarrow \mathsf{at\_risk}(post_1, o), \neg\mathsf{parent}(o, sandisk)$$

$$\omega_2 = \neg\, \mathsf{at\_risk}(post_1, internet\_explorer) \leftarrow \mathsf{at\_risk}(post_1, apple), \neg\mathsf{parent}(apple, internet\_explorer)$$

Δ :

$$\delta_1 = \mathsf{at\_risk}(post_1, microsoft) \prec \mathsf{at\_risk}(post_1, o), \mathsf{parent}(o, microsoft)$$

$$\delta_2 = \mathsf{at\_risk}(post_1, safari) \prec \mathsf{at\_risk}(post_1, apple), \mathsf{parent}(apple, safari)$$

$$\delta_3 = \mathsf{at\_risk}(post_1, apple) \prec \mathsf{user\_preference}(userID_1, apple)$$

$$\delta_4 = \mathsf{at\_risk}(post_1, o) \prec \mathsf{previously\_seen}(webID_1, o)$$

Fig. 6.4: A ground argumentation framework.

$\langle \mathcal{A}_1, \mathsf{at\_risk}(post_1, microsoft) \rangle$	$\mathcal{A}_1 = \{\delta_1, \delta_4, \theta_3\}$
$\langle \mathcal{A}_2, \mathsf{at\_risk}(post_1, safari) \rangle$	$\mathcal{A}_2 = \{\delta_2, \delta_3, \theta_4\}$
$\langle \mathcal{A}_3, \mathsf{at\_risk}(post_1, apple) \rangle$	$\mathcal{A}_3 = \{\delta_3, \theta_5\}$
$\langle \mathcal{A}_4, \mathsf{at\_risk}(post_1, o) \rangle$	$\mathcal{A}_4 = \{\delta_4, \theta_6\}$

Fig. 6.5: Example ground arguments from Fig. 3.2.

6.5 Experiments

We frame the identification of at-risk systems as a multi-label classification problem for each of the system component (platform, vendor, and product)—the basic step involves extracting textual features from the discussions to be used as input to the machine learning models. We now describe the data pre-processing steps and the standard machine learning approaches, along with the metrics used for evaluating the models.

6.5.1 Data Representation

As mentioned above, we use text-based features to represent the hacker discussions on the D2web, which are then used as input to the machine learning models. Some of the discussions are in foreign languages (cf. Fig. 6.2). The commercial data collection platform automatically identifies the language and translates it to English using the Google Translate API [46]. The following pre-processing steps are taken to address different challenges. We employ two feature engineering techniques namely TF-IDF and Doc2Vec.

Text Cleaning. We remove all non-alphanumeric characters from hacker discussions. This removes any *special characters* that do not contribute towards making the decision.

Misspellings and Word Variations. Misspellings and word variations are frequently observed in the discussions on the D2web, leading to separate features in the feature vector if a standard bag-of-words (BOW) approach is used. In BOW, we create a dictionary of

all the word occurrences in the training set; then, for a particular discussion, the feature vector is created by looking up which words have occurred and their count in the discussion. Misspellings and word variations will thus be represented as different words; to address this, we use character n-gram features. As an example, consider the word "execute"—if we were using tri-gram character features, the word "execute" would yield the set of features:

$$\{\textit{“exe”}, \textit{“xec”}, \textit{“ecu”}, \textit{“cut”}, \textit{“ute”}\}.$$

The benefit of this technique is that the variations or misspellings of the word, such as "execution", "executable", or ""exxecute", will all have common features. We found that using character n-grams in the range 3–7 worked best in our experiments.

TF-IDF Features. We vectorize the n-gram features using the term frequency-inverse document frequency (TF-IDF) model, which creates a vocabulary of all the n-grams in the discussion. In TF-IDF, the importance of an n-gram feature increases with the number of times it occurs, but is normalized by the total number of n-grams in the description. This eliminates common words from being important features. We consider the top 1,000 most frequent features (using more than 1,000 features did not improve the performance, but rather only added to the training and testing time).

Doc2Vec Features. Doc2Vec is a feature engineering technique to generate document vector (in our case document refers to a discussion), which acts as input to the classifier to identify at-risk systems. In Doc2Vec, first, a vector representation of each word in the document in computed by taking into account the words around it (to maintain context) and then these word vectors are averaged to get a representation of the document. We implement Doc2Vec using the *gensim* library in Python [4] . It was been previously used to classify tweets [128] as well as product descriptions [66].

[4]https://radimrehurek.com/gensim/models/doc2vec.html

6.5.2 Supervised Learning Approaches

We conducted our experiments using the following standard machine learning approaches implemented using Python machine learning library [5] discussed in Section 2.3.

6.5.3 Evaluation Metrics

In our experiments, we evaluate performance based on three metrics: *precision*, *recall*, and *F1 measure*. For a given hacker discussion, precision is the fraction of labels (platforms, vendors, or products) that the model associated with the discussion that were *actual labels* in the ground truth. Recall, on the other hand, is the fraction of ground truth labels *identified* by the model. The F1 measure is the harmonic mean of precision and recall. In our results, we report the average precision, recall, and F1 for all the test discussions.

6.5.4 Baseline Model (BM)

For the baseline model, we only leverage the machine learning technique to identify the at-risk systems. We create training and testing sets by sorting the discussions by posted time on the website (to avoid temporal intermixing). We reserve the first 80% of the samples for training and the rest (20%) for testing. We employed both TF-IDF and Doc2Vec as feature engineering techniques. On conducting the experiments, it was observed that TF-IDF performed better than Doc2Vec in all the experiments. Hence we only report the results using TF-IDF features.

Results. Table 6.4 shows the average performance of the machine learning technique for each component of the at-risk system. For *platform* identification, SVM performs the best with the following averages:

- *precision*: 0.72,

[5] http://scikit-learn.org/stable/

Table 6.4: Average Precision, Recall, and F1 measure for NB, LOG-REG, DT, RF and SVM to identify at-risk systems.

Component	**Model**	**Precision**	**Recall**	**F1 measure**
Platform	NB	0.68	0.65	0.66
	LOG-REG	**0.72**	0.76	0.74
	DT	0.66	0.70	0.68
	RF	0.70	0.75	0.72
	SVM	**0.72**	**0.78**	**0.76**
Vendor	NB	0.37	0.34	0.36
	LOG-REG	0.28	0.25	0.27
	DT	0.39	0.43	0.41
	RF	**0.40**	0.43	0.41
	SVM	**0.40**	**0.48**	**0.44**
Product	NB	0.19	0.14	0.16
	LOG-REG	0.20	0.13	0.16
	DT	0.22	0.15	0.18
	RF	0.22	**0.25**	0.24
	SVM	**0.26**	0.24	**0.25**

- *recall*: 0.78, and
- *F1 measure*: 0.76.

LOG-REG had similar precision, but lower recall. Similarly, for vendor identification, SVM performs the best with averages:

- *precision*: 0.40,
- *recall*: 0.48, and

- *F1 measure*: 0.44,

with RF having similar precision. For *platform* identification, SVM had the best performance:

- *precision*: 0.28,
- *recall*: 0.24 (comparable to RF), and
- *F1 measure*: 0.25.

Since SVM performs consistently better for all three classification problems, moving forward we use SVM as our machine learning component in the reasoning framework (cf. Fig. 6.1).

6.5.5 Reasoning Framework (RFrame)

As we go down the CPE hierarchy, the number of possible labels for vendors and products increases largely as the number of discussions representing each label decreases, thus making learning difficult and decreasing performance. We address this issue by proposing a set of strict and defeasible rules for platform, vendor, and product identification. We note that these rules arise from the discussion that is being evaluated and do not require parameter learning.

We use the notation described in Table 6.5 for defining our constructs (facts, strict rules, and defeasible rules). We note that facts cannot have variables, only constants (however, to compress the program for presentation purposes, we use *meta-variables* in facts). To begin, we define the facts (see Fig. 6.6): θ_1 states that a hacker discussion $\mathcal{D}$ was posted on the D2web website $\mathcal{W}$ (can be either forum or marketplace), and θ_2 states that the user $\mathcal{U}$ posted the discussion. For each level in the CPE hierarchy, we define additional rules discussed as follows.

Table 6.5: Notation and Explanations

Notation	**Explanation**
$\mathcal{D}$	The hacker discussion (posted on the website) under consideration.
$\mathcal{W}$	Website (marketplace or forum) where the hacker discussion was posted.
$\mathcal{S}_w$, $\mathcal{V}_w$ and $\mathcal{P}_w$	The set of platforms, vendors and products at-risk by the hacker discussions previously seen in $\mathcal{W}$ under consideration respectively.
$\mathcal{U}$	User posting the hacker discussion.
$\mathcal{S}_u$, $\mathcal{V}_u$ and $\mathcal{P}_u$	The set of platforms, vendors and products at-risk by the hacker discussions previously posted by user $\mathcal{U}$ under consideration respectively.
$\mathcal{S}_p$, $\mathcal{V}_p$ and $\mathcal{P}_p$	The set of platforms, vendors and products identified by the machine learning model at each level in the hierarchy for hacker discussions under consideration respectively.
s_i, v_i and p_i	Each element of the set $\mathcal{S}_p$, $\mathcal{V}_p$ and $\mathcal{P}_p$ representing a single platform, vendor or product respectively.

$$\Theta: \quad \theta_1 = \mathsf{posted}(\mathcal{D}, \mathcal{W}) \qquad \theta_2 = \mathsf{posted}(\mathcal{D}, \mathcal{U})$$

Fig. 6.6: Facts defined for each test discussion.

Platform Model. The first level of system identification is identifying the platform that the hacker discussion is a threat to. We compute previously discussed platforms on D2web websites under consideration. Similarly, which platform the user under consideration prefers (based on their previous postings) is also computed. This shows preferred platform discussions on websites and by users, which can aid the machine learning model in reducing

$$
\begin{array}{lll}
 & & \text{For } s \in \mathcal{S}_w\text{:} \\
\Delta: & \delta_1 = & \mathsf{at\_risk}(\mathcal{D}, s) \prec \mathsf{previously\_seen}(\mathcal{W}, s). \\
 & & \text{For } s \in \mathcal{S}_u\text{:} \\
 & \delta_2 = & \mathsf{at\_risk}(\mathcal{D}, s) \prec \mathsf{user\_preference}(\mathcal{U}, s).
\end{array}
$$

Fig. 6.7: Defeasible rules for platform identification.

the number of platforms it can identify from. The DeLP components that model platform identification are shown in Fig. 6.7. For the defeasible rules, δ_1 indicates that all the platforms $\mathcal{S}_w$ previously seen in the D2web website $\mathcal{W}$ where the current discussion $\mathcal{D}$ is observed are likely at-risk, δ_2 indicates that all the platforms $\mathcal{S}_u$ from user $\mathcal{U}$'s previous postings are also likely at-risk.

Vendor Model. The second level is identifying the at-risk vendor. For this case, we use the platform result from the previous model, taking that as a DeLP *fact*. The DeLP components that model vendor identification are shown in Fig. 6.8. Here, the fact θ_1 indicates the platform identified for the discussion—note that multiple platforms may be identified based on the discussion. The strict rule ω_1 states that for a given post $\mathcal{D}$ posing a threat to platform s, the vendor v_i cannot be at-risk if the parent of v_i is not the identified platform s. This rule is based on the CPE hierarchy obtained from NVD. For the defeasible rules, δ_1 indicates that all the vendors Y_w previously seen in the D2web website $\mathcal{W}$ where the current hacker discussion $\mathcal{D}$ is observed are likely at-risk, δ_2 indicates that all the vendors Y_u from user $\mathcal{U}$'s previous postings are also likely at-risk, and δ_3 states that for a given post $\mathcal{D}$ posing a threat to platform s, all the vendors whose parent is the identified platform are likely at-risk. This rule is also based on the CPE hierarchy from NVD.

Product Model. The third level is identifying the at-risk product. For this case, we use the vendor result from the previous model; as before, we use that as a DeLP *fact*. The

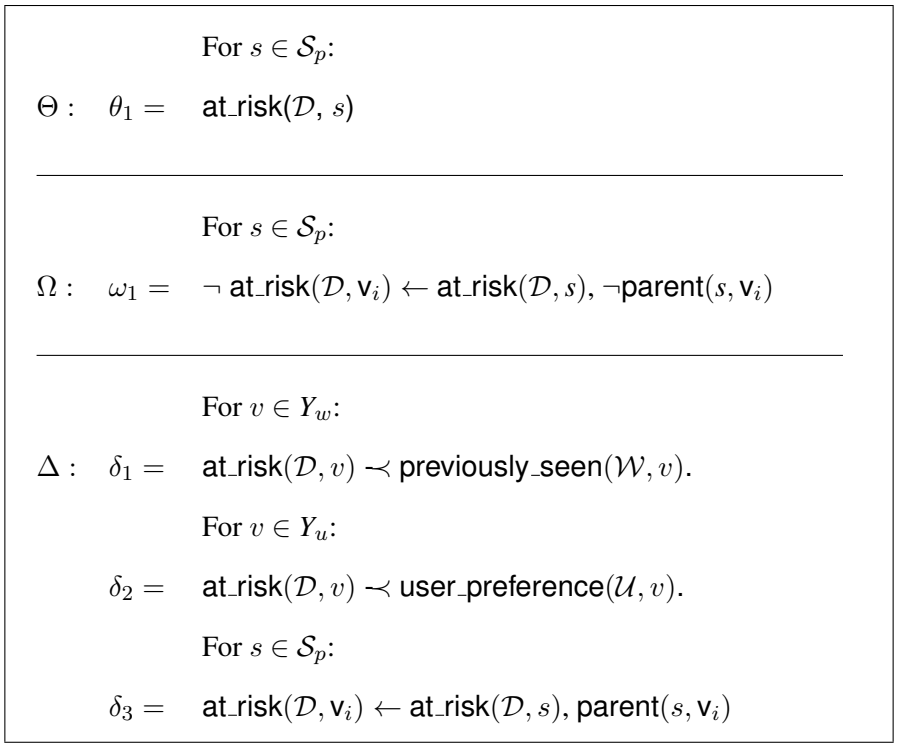


Fig. 6.8: Defeasible rules for vendor identification.

DeLP components that model product identification are shown in Fig. 6.9. Here, the fact θ_1 indicates the vendor identified for the discussion—again, multiple vendors may be identified based on the discussion. The strict rule ω_1 states that for a given post $\mathcal{D}$ posing a threat to vendor v, the product p_i cannot be at-risk if the parent of p_i is not the identified vendor v (again, based on the CPE hierarchy). For the defeasible rules, δ_1 indicates that all the products $\mathcal{P}_w$ previously seen in the D2web website $\mathcal{W}$ where the current hacker discussion $\mathcal{D}$ is observed are likely at-risk, δ_2 indicates that all the products $\mathcal{P}_u$ from user $\mathcal{U}$'s previous postings are also likely at-risk, and δ_3 states that for a given post $\mathcal{D}$ posing a threat to vendor v, all the products whose parent (in the CPE hierarchy) is the identified vendor are likely at-risk.

Results. We evaluate the reasoning framework using an experimental setup similar to the one discussed in the baseline model. We report the precision, recall, and F1 measure for

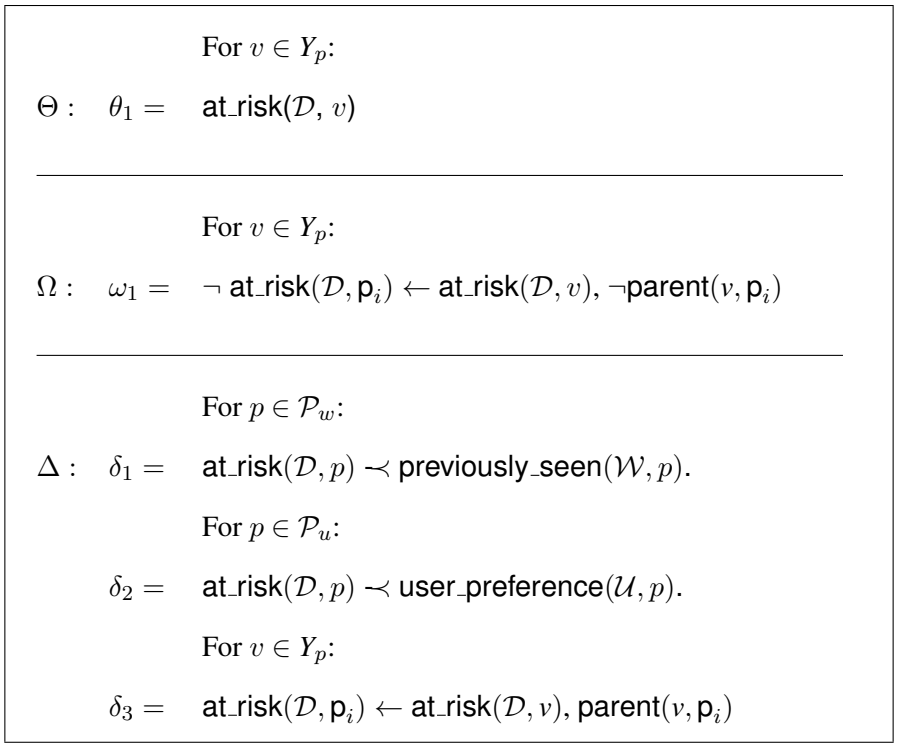


Fig. 6.9: Defeasible rules for product identification.

each of the system components and compare them with the best performing baseline model (BM). Table 6.6 shows the comparison between the two models.

For *platform* identification, RFrame outperforms BM in terms of precision: 0.83 vs. 0.72 (a 15.27% improvement), while maintaining the same recall. Similarly, for *vendor* and *product* identification there was significant improvement in precision: 0.56 vs. 0.40 (a 40% improvement) and 0.41 vs. 0.26 (a 57.69% improvement), respectively, with comparable recall with respect to the baseline model. The major reason for the jump in precision is the reduction of possible labels based on the arguments introduced that aids the machine learning model to make the correct decision.

Table 6.6: Average Precision, Recall, and F1 measure comparison between the baseline model (BM) and reasoning framework (RFrame).

Component	**Model**	**Precision**	**Recall**	**F1 measure**
Platform	BM	0.72	**0.78**	0.76
	RFrame	**0.83**	**0.78**	**0.80**
Vendor	BM	0.40	**0.48**	0.44
	RFrame	**0.56**	0.44	**0.50**
Product	BM	0.26	**0.24**	0.25
	RFrame	**0.41**	0.21	**0.30**

6.6 Discussion

The performance of the reasoning system highlights that our hybrid framework identifies at-risk systems with higher precision with respect to the approach using only machine learning classifiers. In our application, we desire a high precision—while maintaining at least comparable recall—in order to provide high value risk assessment of systems; low precision is often equated to a less reliable framework. The majority of misclassifications are a result of less data representing those systems in the training set; for some system components, the instances can be as low as having only one discussion in the training set. This issue becomes more relevant as we go down the hierarchy with large numbers of vendors and products. In some test instances, for the same platform and vendor, a new product not previously known to be at-risk becomes vulnerable due to a newly disclosed vulnerability. In this case, the reasoning framework is not able to identify the product since it was not previously observed, and this can contribute to a misclassification.

From a security analyst's perspective, the reasoning framework not only provides a list of possible at-risk systems but also provides arguments indicating *why* a particular system was

identified as being at-risk. This lets the analyst evaluate the decisions made by the framework and fine-tune it if necessary. For cases where a new product (not previously discussed in training) is at-risk, even a partial identification of the system (in terms of platform and vendor) is of value to the analyst. Based on the alert provided by the framework, the analyst can manually evaluate the arguments and the discussions to identify possible products, depending on the platform and vendor identified by the framework.

6.7 Related Work

Threat assessment of systems is critical to organizations' security policy. Over the years, CVSS [33] has become a standard metric that organizations use to determine if their systems are at risk of being targeted by hackers. Unfortunately, case studies have shown poor correlation between the CVSS score and which system are at-risk [2].

Identifying targeted systems through open source intelligence. Open source intelligence has been used previously to identify and predict vulnerabilities that are likely to be exploited to determine which systems are at risk. [138] has looked to predict the likelihood that a software has a vulnerability not yet discovered using the national vulnerability database (NVD). They show that NVD has a poor prediction capability in doing so due to limited amount of information available. On the other hand, [106] looks to predict if a real world exploit is available based on vulnerabilities disclosed from Twitter data. The authors report high accuracies of 90% using a resampled, balanced, and temporal mixed dataset, not reflective of real world scenarios [21]. Identifying threats to critical infrastructure by analyzing interactions on hacker forums was studied in [76]. Here the authors reply on keyword based queries to identify such threats from hacker interactions. Tools to automatically identify products offered in cyber criminal markets was proposed in [100]. This technique looks to extract products mentioned in the description of the item that is being offered, a problem different than what we address – identifying targeted systems not

explicitly stated in the forum discussions.

More recently, researchers have shown increased interest on gathering threat intelligence from D2web to pro-actively identify digital threats and study hacker communities to gather insights. Researchers have focused on building infrastructure to gather threat information from markets (regarding goods and services sold) and forums (discussions regarding exploits and) [90, 103], studying the different product categories offered in darkweb markets – creating a labeled dataset [79], analyzing hacker forums and carding shops to identify potential threats [10], identify expert hackers to determine their specialties [1], identify key hackers based on posted content, their network and since when they are active in the forum [80]. For vulnerability research, studies look to leverage vulnerability mentions in the D2web to predict the likelihood of exploitation using a combination of machine learning and social network techniques [4, 3]. These techniques rely on the mentions of CVE numbers to identify likely targeted systems (which is a small fraction of vulnerabilities [4]), not taking into account discussions where a CVE number is not mentioned. On the other hand, we look to identify the at-risk systems *without having a CVE number*, which is a different problem from those tackled in previous work.

Identifying targeted systems through software analysis. Another way of identifying targeted softwares with vulnerabilities deals with analyzing the *software itself* in order to determine which component of the software is most likely to contain a vulnerability. Mapping past vulnerabilities to vulnerable software components was proposed in [85], where the authors found that components with function calls and import statements are more likely to have a vulnerability. A similar method was employed by [111, 131], where text mining was used to forecast whether a particular software component contains vulnerabilities. Similar text mining techniques for vulnerability discovery are listed in [43]. The text mining methods create a count dictionary of terms used in the software, which are used as features to identify vulnerabilities. These methods suffer from the issue of not knowing which

vulnerabilities might be of interest to hackers. On the other hand, we work with hacker discussions posing a threat to systems that are of clearly of interest to hackers since they are discussing them on the D2web websites—vulnerabilities mentioned on D2web regarding systems are more likely to be exploited [4].

6.8 Summary

In this chapter, we demonstrated how a reasoning framework based on the DeLP structured argumentation system can be leveraged to improve the performance of identifying at-risk systems based on hacker discussions on the D2web. DeLP programs built on discussions found on forums and marketplaces afford a reduction on the set of possible platforms, vendors, and products that are likely to be at-risk by the hackers participating in the discussion. This reduction of potential labels leads to better precision while almost maintaining comparable recall as compared to the baseline model that only leverages machine learning techniques. Knowing discussed systems by threat actors as possible targets helps organizations achieve better threat assessment for their systems.

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this dissertation, we described three pieces of work to better understand and reason about the activities of cyber threat actors. This is achieved by covering three factors: the threat actor's use of deception, the capabilities available, and the intent of launching the attack.

In *Chapter 2*, we considered the problem of cyber-attribution by examining DEFCON CTF data - which provides us with ground-truth on the culprit responsible for each attack. We frame cyber-attribution as a classification problem and examine it using several machine learning approaches. We evaluated the approaches on nearly 10 million attacks from the CTF data. Random forest achieved the best performance of identifying 37% of the attacking teams. We find that deceptive attacks – where same exploits are used by multiple teams to target a particular team, account for the majority of misclassified samples. The usage of the same exploit creates similar feature representations for different attacking teams - making it difficult for machine learning approaches. We employ several pruning techniques to alleviate the misclassification due to deception.

In *Chapter 3*, we considered the problem of misclassification in a cyber attribution scenario introduced in Chapter 2. The main source of misclassification was the deceptive attacks. In this chapter we demonstrate how leveraging Defeasible Logic Programming (DeLP) in an argumentation-based framework, can be employed to improve cyber-attribution decisions. This is done by building DeLP programs based on real-world data; this approach affords a reduction of the set of potential culprits and thus greater accuracy when using a

classifier for cyber attribution. We thus proposed a hybrid system that integrates argumentation with a machine learning model to make decisions. Using this reasoning framework the accuracy of identifying the attacking team jumped from 37% to 64.5% – a significant improvement. The framework also aided a security analyst by providing a set of arguments as to why a particular team was identified as the attacker and other teams were not.

In *Chapter 4*, we considered the problem of determination of adversarial intent (tasks) on the attacked system by analyzing the malware/exploit used in the attack. Specifically, identifying the tasks (intent) a given piece of malware was designed to perform (e.g., logging keystrokes, recording video, establishing remote access). We present an automated method to identify malware tasks using two different approaches based on the ACT-R cognitive architecture, a popular implementation of a unified theory of cognition. Using three different malware collections, we explore various evaluations for each of an instance-based and rule-based model - including cases where the training data differs significantly from test; where the malware being evaluated employs packing to thwart analytical techniques; and conditions with sparse training data. Features are constructed by combining both static and dynamic malware analysis – considering the function call names, network activity, malware behavior on the host machine and so on. We find that our approach based on cognitive inference consistently out-performs the current state-of-the art software for malware task identification as well as standard machine learning approaches - often achieving an unbiased F1 score of over 0.9. We also show the scaling of the model in terms of test time as the number of samples grow.

In *Chapter 5*, we considered the problem of gathering intelligence related to malicious hacking. We consider social platforms on darkweb and deepweb – in particular hacker forums and marketplaces, for data collection. We address various design challenges to develop a focused crawler using data mining and machine learning techniques resulting in an operational system for identifying emerging cyber threats. At the time of development the

system was actively collecting approximately 305 cyber threats each week. Since then, we have transitioned this system to a commercial partner to increase the scale of data collection and maintenance These threat warnings include information on newly developed malware and exploits that have not yet been deployed in a cyber-attack, discussions regarding known vulnerabilities and how they can be exploited, users having presence in multiple platforms to get better understanding of their social connections. This provides a significant service to cyber-defenders. The system is significantly augmented through the use of various data mining and machine learning techniques. With the use of learning models, we are able to recall 92% of products in marketplaces and 80% of discussions on forums relating to malicious hacking with high precision.

In *Chapter 6*, we considered the problem of identifying systems likely to be at-risk by threat actors to help organizations better defend against likely cyber attacks. We leverage hacker discussions on darkweb marketplaces and forums collected using the system introduced in Chapter 5 to identify the platforms, vendors, and products likely to be at-risk by hackers. This gives us an indicator regarding the hacker capability of targeting systems based on their discussions. We employed and modified the reasoning system introduced in Chapter 3 that combines DeLP (Defeasible Logic Programming) and machine learning classifiers to identify systems based on hacker discussions observed on the darkweb. The modified system takes into account the hierarchical structure of identifying a system in terms of its platform, vendor and product. The system is evaluated on hacker discussions from nearly 300 darkweb forums and marketplaces. We improved precision by 15%–57% while maintaining recall over baseline approaches.

7.2 Future Work

There are a number of interesting future directions for research. Some specific future work that extends the work in different chapters is discussed below.

In *Chapter 3*, we presented a reasoning framework to handle contradictory and inconsistent evidence resulting in cases of deceptive attacks to identify threat actors. In our experiments, arguments are defeated based on contradicting information in other arguments without any preference in terms of confidence in the arguments being defeated. A probabilistic variant of DeLP [114], can result in a preference list to decide defeat of an argument in case of contradiction.

In addition to understanding and identifying threat actors, we also ask: how do researchers train and evaluate frameworks? The DEFCON CTF competition is designed to make the incentive structure match the real world: contestants only receive points when they hack into a system. The one thing teams in the CTF competition are not concerned with is being discovered by other teams. The goal is to exploit vulnerabilities and score as many points as possible. There is no incentive in the form of extra points awarded for being deceptive where the target team cannot identify the adversary. In real world attack scenarios where deception is an important goal, attackers employ different strategies to protect their identity i.e. masking their IP address, using other systems as decoy to launch their attacks and so on. There might me instances where such behavior occurs in the CTF competition (see Chapter 2) but is not the priority with no incentives attached. To study and encourage such behavior we designed our own CTF competition recently presented in [105]. By motivating the contestants to employ deception, the data we gather will be more relevant to studying deception in attribution while retaining ground truth. As the game masters we can maintain visibility of the true facts of the game, and we can contrast contestant performance with the performance of algorithms developed for the purpose of countering deception. The framework is also capable of capturing host-level interactions that occur in the context of the vulnerable running program. We make our platform available as open-source software. It can be downloaded from: https://github.com/trailofbits/attribution-vm.

In *Chapter 4*, we analyze malicious samples by running them inside a sandbox to capture

the behavior on the host machine in terms of network and file activity. There are cases of highly-sophisticated malware that in addition to using encryption and packing to limit static analysis, also employ methods to "shut down" when run in a sandbox environment [72]. Methods to address such cases such as the technique of "spatial analysis" [44] that involves direct analysis of a malware binary can he helpful.

REFERENCES

[1] A. Abbasi, W. Li, V. Benjamin, S. Hu, and H. Chen. Descriptive analytics: Examining expert hackers in web forums. In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, pages 56–63. IEEE, 2014.

[2] L. Allodi and F. Massacci. Comparing vulnerability severity and exploits using case-control studies. *ACM Transactions on Information and System Security (TISSEC)*, 17(1):1, 2014.

[3] M. Almukaynizi, A. Grimm, E. Nunes, J. Shakarian, and P. Shakarian. Predicting cyber threats through the dynamics of user connectivity in darkweb and deepweb forums. In *ACM Computational Social Science*. ACM, 2017.

[4] M. Almukaynizi, E. Nunes, K. Dharaiya, M. Senguttuvan, J. Shakarian, and P. Shakarian. Proactive identification of exploits in the wild through vulnerability mentions online. In *2017 International Conference on Cyber Conflict (CyCon U.S.)*, pages 82–88, Nov 2017.

[5] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of mind. *PSYCHOLOGICAL REVIEW*, 111:1036–1060, 2004.

[6] C. Annachhatre, T. H. Austin, and M. Stamp. Hidden markov models for malware classification. *Journal of Computer Virology and Hacking Techniques*, 11(2):59–73, 2015.

[7] A. Applebaum, K. Levitt, Z. Li, S. Parsons, J. Rowe, and E. Sklar. Cyber reasoning with argumentation: Abstracting from incomplete and contradictory evidence. In *Proc. of MILCOM*, 2015.

[8] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering, 2009.

[9] M. Belkin and P. Niyogi. Using manifold structure for partially labelled classification. In *Advances in NIPS*, 2002.

[10] V. Benjamin, W. Li, T. Holt, and H. Chen. Exploring threats and vulnerabilities in hacker web: Forums, irc and carding shops. In *Intelligence and Security Informatics (ISI), 2015 IEEE International Conference on*, pages 85–90. IEEE, 2015.

[11] C. M. Bishop and I. Ulusoy. Object recognition via local patch labelling. In *Deterministic and Statistical Methods in Machine Learning*, pages 1–21, 2004.

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[13] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100, New York, NY, USA, 1998. ACM.

[14] W. E. Boebert. A survey of challenges in attribution. In *Proceedings of a workshop on Deterring CyberAttacks*, pages 41–54, 2010.

[15] D. Bothell. Act-r 6.0 reference manual. `http://act-r.psy.cmu.edu/actr6/reference-manual.pdf`, 2004.

[16] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–114. ACM, 2010.

[17] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[18] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[19] M. Brückner, C. Kanzow, and T. Scheffer. Static prediction games for adversarial learning problems. *The Journal of Machine Learning Research*, 13(1):2617–2654, 2012.

[20] M. Brückner and T. Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 547–555. ACM, 2011.

[21] B. L. Bullough, A. K. Yanchenko, C. L. Smith, and J. R. Zipkin. Predicting exploitation of disclosed software vulnerabilities using open-source data. In *Proceedings of the 2015 ACM International Workshop on International Workshop on Security and Privacy Analytics*. ACM, 2017.

[22] J. Carr. *The Evolving State of Cyber Warfare*. Project Grey Goose, 2009.

[23] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th international conference on World Wide Web*, pages 148–159. ACM, 2002.

[24] S. Chakrabarti, M. Van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11):1623–1640, 1999.

[25] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[26] H. Chen. *Dark web: Exploring and data mining the dark side of the web*, volume 30. Springer Science & Business Media, 2011.

[27] H. Cheng, Z. Liu, and J. Y. 0001. Sparsity induced similarity measure for label propagation. In *ICCV*, pages 317–324. IEEE, 2009.

[28] I. K. Cho and E. G. Im. Extracting representative api patterns of malware families using multiple sequence alignments. In *Proceedings of the 2015 Conference on research in adaptive and convergent systems*, pages 308–313. ACM, 2015.

[29] J. B. M. S. Claudio Guarnieri, Alessandro Tanasi. Cuckoo sandbox. `http://www.cuckoosandbox.org/`, 2012.

[30] C. Cortes and V. Vapnik. Support-vector networks. pages 273–297, 1995.

[31] CPE. Official common platform enumeration dictionary. https://nvd.nist.gov/cpe.cfm, Last Accessed: Feb 2018.

[32] CVE. Common vulnerabilities and exposures: The standard for information security vulnerability names. http://cve.mitre.org/, Last Accessed: Feb 2018.

[33] CVSS. Common vulnerability scoring system. https://www.first.org/cvss, Last Accessed: Feb 2018.

[34] M. Dacier, V.-H. Pham, and O. Thonnard. The wombat attack attribution method: some results. In *Information Systems Security*, pages 19–37. Springer, 2009.

[35] DEFCON. Defcon: Capture the flag. 2013.

[36] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, 2004.

[37] P. M. Dung. On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

[38] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho. Analysis of machine learning techniques used in behavior-based malware detection. In *Proceedings of the 2010 Second International Conference on ACT*, ACT '10, pages 201–203, Washington, DC, USA, 2010. IEEE Computer Society.

[39] FireEye. Against cyber threats, knowledge is power. https://www.fireeye.com/products/cyber-threat-intelligence.html, Last Accessed: Feb 2018.

[40] Fortinet. Know your vulnerabilities get the facts about your network security. https://www.fortinet.com/assessment, Last Accessed: Feb 2018.

[41] T. Fu, A. Abbasi, and H. Chen. A focused crawler for dark web forums. *Journal of the American Society for Information Science and Technology*, 61(6):1213–1231, 2010.

[42] A. J. García and G. R. Simari. Defeasible logic programming: An argumentative approach. *Theory and practice of logic programming*, 4(1+ 2):95–138, 2004.

[43] S. M. Ghaffarian and H. R. Shahriari. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Computing Surveys (CSUR)*, 50(4):56, 2017.

[44] D. Giametta and A. Potter. There and back again:a critical analysis of spatial analysis, 2014.

[45] C. Gonzalez, J. F. Lerch, and C. Lebiere. Instance-based learning in dynamic decision making. *Cognitive Science*, 27(4):591 – 635, 2003.

[46] Google. Google cloud translation api documentation. https://cloud.google.com/translate/docs/, Last Accessed: Feb 2018.

[47] GVDG. Generator malware gvdg. 2011.

[48] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen. An approach for detection and family classification of malware based on behavioral analysis. In *Computing, Networking and Communications (ICNC), 2016 International Conference on*, pages 1–5. IEEE, 2016.

[49] T. Holt and B. Schell. *Hackers and Hacking: A Reference Handbook*. Contemporary World Issues.

[50] T. J. Holt. Subcultural evolution? examining the influence of on-and off-line experiences on deviant subcultures. *Deviant Behavior*, 28(2):171–198, 2007.

[51] T. J. Holt and E. Lampke. Exploring stolen data markets online: products and market forces. *Criminal Justice Studies*, 23(1):33–50, 2010.

[52] T. J. Holt, D. Strumsky, O. Smirnova, and M. Kilger. Examining the social networks of malware writers and hackers. *International Journal of Cyber Criminology*, 6(1):891–903, 2012.

[53] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[54] Invencia. Crowdsource: Crowd trained machine learning model for malware capability detection. `http://www.invincea.com/tag/cynomix/`, 2013.

[55] ISEC-Lab. Anubis: Analyzing unknown binaries. `http://anubis.iseclab.org/`, 2007.

[56] S. Jajodia, P. Shakarian, V. S. Subrahmanian, V. Swarup, and C. Wang. *Cyber Warfare: Building the Scientific Foundation*. Springer Publishing Company, Incorporated, 2015.

[57] T. Jordan and P. Taylor. A sociology of hackers. *The Sociological Review*, 46(4):757–780, 1998.

[58] H. K. Kalutarage, S. Shaikh, Q. Zhou, A. E. James, et al. Sensing for suspicion at scale: A bayesian approach for cyber conflict attribution and reasoning. In *Cyber conflict (CYCON), 2012 4th international conference on*, pages 1–19. IEEE, 2012.

[59] Kaspersky. Gauss: Abnormal distribution, 2012.

[60] J. Kinable and O. Kostakis. Malware classification based on call graph clustering. *J. Comput. Virol.*, 7(4):233–245, Nov. 2011.

[61] D. Kong and G. Yan. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD*, KDD '13, pages 1357–1365, New York, NY, USA, 2013. ACM.

[62] D. Lacey and P. M. Salmon. It's dark in there: Using systems analysis to investigate trust and engagement in dark web forums. In D. Harris, editor, *Engineering Psychology and Cognitive Ergonomics*, volume 9174 of *Lecture Notes in Computer Science*, pages 117–128. Springer International Publishing, 2015.

[63] C. Lebiere, S. Bennati, R. Thomson, P. Shakarian, and E. Nunes. Functional cognitive models of malware identification. *Proceedings of ICCM, ICCM*, pages 9–11, 2015.

[64] C. Lebiere, S. Bennati, R. Thomson, P. Shakarian, and E. Nunes. Functional cognitive models of malware identification. In *Proceedings of ICCM, ICCM 2015, Groningen, The Netherlands, April 9-11, 2015*, 2015.

[65] C. Lebiere, P. Pirolli, R. Thomson, J. Paik, M. Rutledge-Taylor, J. Staszewski, and J. R. Anderson. A functional model of sensemaking in a neurocognitive architecture. *Intell. Neuroscience*, 2013:5:5–5:5, Jan. 2013.

[66] H. Lee and Y. Yoon. Engineering doc2vec for automatic classification of product descriptions on o2o applications. *Electronic Commerce Research*, pages 1–24, 2017.

[67] A. Levin, D. Lischinski, and Y. Weiss. A closed form solution to natural image matting. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 61–68, Washington, DC, USA, 2006. IEEE Computer Society.

[68] S. Levy. *Hackers: Heroes of the Computer Revolution*. Doubleday, New York, NY, USA, 1984.

[69] B. Li and Y. Vorobeychik. Feature cross-substitution in adversarial classification. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2087–2095. Curran Associates, Inc., 2014.

[70] P. Li, L. Liu, D. Gao, and M. K. Reiter. On challenges in evaluating malware clustering. In *RAID*, volume 6307, pages 238–255. Springer, 2010.

[71] P. Li, L. Liu, and M. K. Reiter. On challenges in evaluating malware clustering, 2007.

[72] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti. Detecting environment-sensitive malware. In *Proceedings of the 14th International Conference on RAID*, RAID'11, pages 338–357, Berlin, Heidelberg, 2011. Springer-Verlag.

[73] J. W. Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[74] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.

[75] M. Macdonald, R. Frank, J. Mei, and B. Monk. Identifying digital threats in a hacker web forum. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 926–933. ACM, 2015.

[76] M. Macdonald, R. Frank, J. Mei, and B. Monk. Identifying digital threats in a hacker web forum. In *Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on*, pages 926–933. IEEE, 2015.

[77] Mandiant. Apt1:exposing one of china's cyber espionage units. `http://intelreport.mandiant.com/`, 2013.

[78] Mandiant. Mandiant APT1 samples categorized by malware families. *Contagio Malware Dump*, 2013.

[79] E. Marin, A. Diab, and P. Shakarian. Product offerings in malicious hacker markets. In *Intelligence and Security Informatics (ISI), 2016 IEEE Conference on*, pages 187–189. IEEE, 2016.

[80] E. Marin, J. Shakarian, and P. Shakarian. Mining key-hackers on darkweb forums. In *International Conference on Data Intelligence and Security (ICDIS), 2018*. IEEE, 2018.

[81] M. Mateski, C. M. Trevino, C. K. Veitch, J. Michalski, J. M. Harris, S. Maruoka, and J. Frye. Cyber threat metrics. *Sandia National Laboratories*, 2012.

[82] J. Mei and R. Frank. Sentiment crawling: Extremist content collection through a sentiment analysis guided web-crawler. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 1024–1027. ACM, 2015.

[83] F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *ACM Transactions on Internet Technology (TOIT)*, 4(4):378–419, 2004.

[84] M. Motoyama, D. McCoy, K. Levchenko, S. Savage, and G. M. Voelker. An analysis of underground forums. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 71–80. ACM, 2011.

[85] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller. Predicting vulnerable software components. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 529–540. ACM, 2007.

[86] NIST. National vulnerability database. https://nvd.nist.gov/, Last Accessed: Feb 2018.

[87] E. Nunes, C. Buto, P. Shakarian, C. Lebiere, S. Bennati, R. Thomson, and H. Jaenisch. Malware task identification: A data driven approach. In *Advances in Social Networks Analysis and Mining (ASONAM), 2015 IEEE/ACM International Conference on*, pages 978–985. IEEE, 2015.

[88] E. Nunes, C. Buto, P. Shakarian, C. Lebiere, R. Thomson, S. Bennati, and J. Holger. Malware task identification : A data driven approach. In *Proceedings of International Symposium on Foundation of Open Source Intelligence and Security Informatics (FOSINT-SI)*. IEEE, 2015.

[89] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath, J. Robertson, J. Shakarian, A. Thart, and P. Shakarian. Darknet and deepnet mining for proactive cybersecurity threat intelligence. In *Intelligence and Security Informatics (ISI), 2016 IEEE Conference on*, pages 7–12. IEEE, 2016.

[90] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath, J. Robertson, J. Shakarian, A. Thart, and P. Shakarian. Darknet and deepnet mining for proactive cybersecurity threat intelligence. In *Proceeding of ISI 2016*, pages 7–12. IEEE, 2016.

[91] E. Nunes, N. Kulkarni, P. Shakarian, A. Ruef, and J. Little. Cyber-deception and attribution in capture-the-flag exercises. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France*, pages 962–965, 2015.

[92] E. Nunes, N. Kulkarni, P. Shakarian, A. Ruef, and J. Little. Cyber-deception and attribution in capture-the-flag exercises. In *Cyber Deception*, pages 149–165. Springer, 2016.

[93] E. Nunes, P. Shakarian, and G. I. Simari. At-risk system identification via analysis of discussions on the darkweb. In *Electronic Crime Research (eCrime), 2018 APWG Symposium on*. IEEE, 2018.

[94] E. Nunes, P. Shakarian, G. I. Simari, and A. Ruef. Argumentation models for cyber attribution. In *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2016, San Fransisco, USA*, 2016.

[95] J. O'Gorman, D. Kearns, and M. Aharoni. *Metasploit: the penetration tester's guide*. No Starch Press, 2011.

[96] T. Parker, M. Sachs, E. Shaw, and E. Stroz. *Cyber adversary characterization: Auditing the hacker mind*. Syngress, 2004.

[97] R. Perdisci et al. Vamo: towards a fully automated malware clustering validity analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 329–338. ACM, 2012.

[98] R. Perdisci and ManChon. Vamo: towards a fully automated malware clustering validity analysis. In *ACSAC*, pages 329–338. ACM, 2012.

[99] C. P. Pfleeger and S. L. Pfleeger. *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

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

[101] I. Rahwan, G. R. Simari, and J. van Benthem. *Argumentation in artificial intelligence*, volume 47. Springer, 2009.

[102] T. Rid and B. Buchanan. Attributing cyber attacks. *Journal of Strategic Studies*, 38(1-2):4–37, 2015.

[103] J. Robertson, A. Diab, E. Marin, E. Nunes, V. Paliath, J. Shakarian, and P. Shakarian. *Darkweb Cyber Threat Intelligence Mining*. Cambridge University Press, 2017.

[104] J. Robertson, V. Paliath, J. Shakarian, A. Thart, and P. Shakarian. Data driven game theoretic cyber threat mitigation. In *IAAI*, 2016.

[105] A. Ruef, E. Nunes, P. Shakarian, and G. I. Simari. Measuring cyber attribution in games. In *Electronic Crime Research (eCrime), 2017 APWG Symposium on*, pages 28–32. IEEE, 2017.

[106] C. Sabottke, O. Suciu, and T. Dumitra. Vulnerability disclosure in the age of social media: exploiting twitter for predicting real-world exploits. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 1041–1056, 2015.

[107] J. Salvit, Z. Li, S. Perumal, H. Wall, J. Mangels, S. Parsons, and E. I. Sklar. Employing argumentation to support human decision making: A user study. In *AAMAS Workshop on Argumentation in Multiagent Systems*, 2014.

[108] S. Samtani, R. Chinn, and H. Chen. Exploring hacker assets in underground forums. In *Intelligence and Security Informatics (ISI), 2015 IEEE International Conference on*, pages 31–36. IEEE, 2015.

[109] K. Sanders and X. Wang. Malware family identification using profile signatures, Oct. 20 2015. US Patent 9,165,142.

[110] S. Sanner, J. R. Anderson, C. Lebiere, and M. C. Lovett. Achieving efficient and cognitively plausible learning in backgammon. 2000.

[111] R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen. Predicting vulnerable software components via text mining. *IEEE Transactions on Software Engineering*, 40(10):993–1006, 2014.

[112] J. Shakarian, A. T. Gunn, and P. Shakarian. Exploring malicious hacker forums. In *Cyber Deception*, pages 259–282. Springer, 2016.

[113] P. Shakarian, J. Shakarian, and A. Ruef. *Introduction to cyber-warfare: A multidisciplinary approach*. Newnes, 2013.

[114] P. Shakarian, G. I. Simari, and M. A. Falappa. Belief revision in structured probabilistic argumentation. In *Foundations of Information and Knowledge Systems*, pages 324–343. Springer, 2014.

[115] P. Shakarian, G. I. Simari, G. Moores, and S. Parsons. Cyber attribution: An argumentation-based approach. In *Cyber Warfare*, pages 151–171. Springer, 2015.

[116] M. Sikorski and A. Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 1 edition, 2012.

[117] E. I. Sklar, S. Parsons, Z. Li, J. Salvit, S. Perumal, H. Wall, and J. Mangels. Evaluation of a trust-modulated argumentation-based interactive decision-making tool. *Autonomous Agents and Multi-Agent Systems*, pages 1–38, 2015.

[118] K. F. Steinmetz. Craft(y)ness: An ethnographic study of hacking. 55(1):125–145, 2015.

[119] F. Stolzenburg, A. J. García, C. I. Chesnevar, and G. R. Simari. Computing generalized specificity. *Journal of Applied Non-Classical Logics*, 13(1):87–113, 2003.

[120] G. Stoneburner, A. Y. Goguen, and A. Feringa. Sp 800-30. risk management guide for information technology systems. 2002.

[121] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[122] J. Swarner. Before WannaCry was unleashed, hackers plotted about it on the Dark Web. 2017. Available at: `http://www.slate.com/blogs/future_tense/2017/05/23/before_wannacry_was_unleashed_hackers_plotted_about_it_on_the_dark_web.html`.

[123] A. Tamersoy, K. Roundy, and D. H. Chau. Guilt by association: Large scale malware detection by mining file-relation graphs. In *Proceedings of the 20th ACM SIGKDD*, KDD '14, pages 1524–1533. ACM, 2014.

[124] R. Thomson, C. Lebiere, S. Bennati, P. Shakarian, and E. Nunes. Malware identification using cognitively-inspired inference. *Proceedings of BRIMS, BRIMS*, 2015.

[125] R. Thomson, C. Lebiere, S. Bennati, P. Shakarian, and E. Nunes. Malware identification using cognitively-inspired inference. In *Proceedings of BRIMS, BRIMS 2015, Washington DC, March 31-April 3, 2015*, 2015.

[126] O. Thonnard, W. Mees, and M. Dacier. On a multicriteria clustering approach for attack attribution. *ACM SIGKDD Explorations Newsletter*, 12(1):11–20, 2010.

[127] T. Townsend and J. McAllister. Implementation framework-cyber threat prioritization. *Software Engineering Institute, Carnegie Mellon University*, 2013.

[128] L. Q. Trieu, H. Q. Tran, and M.-T. Tran. News classification from social media using twitter-based doc2vec model and automatic query expansion. In *Proceedings of the Eighth International Symposium on Information and Communication Technology*, pages 460–467. ACM, 2017.

[129] N. Tsagourias. Nicolas politis initiatives to outlaw war and define aggression, and the narrative of progress in international law. *European Journal of International Law*, 23(1):255–266, 2012.

[130] S. Turkle. *The Second Self: Computers and the Human Spirit*. Simon & Schuster, Inc., New York, NY, USA, 1984.

[131] J. Walden, J. Stuckman, and R. Scandariato. Predicting vulnerable components: Software metrics vs text mining. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, pages 23–33. IEEE, 2014.

[132] R. J. Walls. *Inference-based Forensics for Extracting Information from Diverse Sources*. PhD thesis, University of Massachusetts Amherst, 2014.

[133] C. Wang, S. Yan, L. Z. 0001, and H.-J. Zhang. Multi-label sparse coding for automatic image annotation. In *CVPR*, pages 1643–1650. IEEE, 2009.

[134] W. Wei. Hunting russian malware author behind phoenix exploit kit. April 2013.

[135] B. Westlake, M. Bouchard, and R. Frank. Assessing the validity of automated webcrawlers as data collection tools to investigate online child sexual exploitation. *Sexual abuse: a journal of research and treatment*, page 1079063215616818, 2015.

[136] D. A. Wheeler and G. N. Larsen. Techniques for cyber attack attribution. Technical report, Institute for Defense Analyses, 2003.

[137] T. J. Wong, E. T. Cokely, and L. J. Schooler. An online database of act-r parameters: Towards a transparent community-based approach to model development. 2010.

[138] S. Zhang, D. Caragea, and X. Ou. An empirical study on using the national vulnerability database to predict software vulnerabilities. In *International Conference on Database and Expert Systems Applications*, pages 217–231. Springer, 2011.

[139] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, pages 58–65, 2003.