Unifying Public Threat Knowledge for Cyber Hunting

by

Michal Shlapentokh-Rothman

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 18, 2020

Certified by .....
Erik Hemberg
Research Scientist
Thesis Supervisor

Certified by .....
Una-May O'Reilly
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Unifying Public Threat Knowledge for Cyber Hunting

by

Michal Shlapentokh-Rothman

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

A common cyber security method is to continuously monitor data flowing into a network. However, the large amount of data that is produced from this approach is quite overwhelming because cyber analysts are unable to review the data in a timely matter. Because of this issue, current security guidelines recommend that organizations pro-actively secure their systems from cyber attacks.There is a large amount of public threat data available for organizations to use as part of their pro-active approaches. Despite the amount of data, there is no consistent collection of such threat data. There also has been little analysis of the public threat data to see how comprehensive it is. Accurate public threat data combined with a security practice such as sensor placement can create a strong active security system. We present two systems, **BRON**, a tool for unifying public threat knowledge, and **CHUCK** (Cyber Hunting Using Public Knowledge), a utility that determines ideal sensor placement using **BRON**. **BRON** is a relational schema that provides easy access to different levels of threat data as well as an analysis of the existing data. **CHUCK** is a system that incorporates **BRON** with co-evolutionary algorithms to identify locations for sensor placement. In this thesis, we first demonstrate how **BRON** can both aid in finding specific threats for a given network and evaluate the quality of threat data. Then we show how **CHUCK**, which uses **BRON**, can identify ideal locations in a network for sensor placement.

Thesis Supervisor: Erik Hemberg
Title: Research Scientist

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist

Acknowledgments

I would like to give a huge thank you to Una-May O'Reilly and Erik Hemberg for all of their support and guidance in both my research and academic endeavors as well as creating an exciting research environment. I would like to thank Bryn Reinstadler for all of her work, ideas, and guidance on the **BRON** part of my thesis. I would like to thank Nick Rutar from Perspecta Labs for all of his assistance and expertise on both **BRON** and **CHUCK**. I would like to thank Nicole for all the wonderful scheduling and administrative work that she does. I would also like to thank other members of the ALFA Lab for their enjoyable company at lab lunches. Lastly, I would like to thank my family for all of their time and constant support throughout my 5 years at MIT.

This material is based upon work supported by the DARPA Advanced Research Project Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contract No. N66001-18-C-4036.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cyber security has become an increasingly dangerous threat for hospitals. A common target for hackers are medical devices such as MRI machines, X-Rays, etc that are usually running old legacy software. Hackers use such devices to gain entry into hospital networks. From there, the hacker can target large parts of a hospital resulting in the loss of billions of dollars, and patient privacy and care[30]. In 2016, a hacker gained access to a hospital network and prevented hospital staff from musing various medical devices unless the hospital paid $17,000 in bitcoin. [43].

Ideally, hospitals would be able to upgrade all of the devices in a hospital. However, patching legacy software can be quite costly and potentially impossible. Alternative methods like intrusion detection [37], require monitoring the data of behavior of numerous devices. Such monitoring results in large amounts of information coming into a system every day. The influx of data that comes from monitoring devices has been an issue for security observation centers [14, 6]. It is extremely difficult to comb through all of the data indiscriminately and search for possible malicious techniques and actions. It is even more difficult, to find a *sequence* of possible techniques and actions that could comprise an attack and indicate that a multi-stage, advanced persistent threat is active.

Given these challenges, cyber analysts are opting to additionally practice '*active cyber security*'. *Active cyber security* involves 'proactively stopping attackers' rather than only defending a network by setting up a perimeter and reacting to alarms

and breaches [17]. In *active cyber security*, rather than process system observations unselectively, specific known attack technique and patterns are consulted and then matched to specific vulnerabilities or attacker-desirable properties of the network. This supports putting targeted sensing in place that allows for specific, selective queries on observations which look for the signatures of the technique. To target such sensing, specific nodes and applications must be identified as vulnerable and sensors placed on them to collect observations of the activity within which the technique is apparent [26]. Examples of such targeted sensing include earmarking an application and configuring it in verbose mode, capturing the traffic flows of a specific router and filtering them for specific flow patterns, or a placing a physical tap on a wire for deep packet inspection.

To practice *active cyber security* cyber analysts rely on known information about attack techniques and patterns. There is an overwhelmingly large amount of information available for cyber analysts to access. This information may include network-specification information or lists of different attacker strategies and tactics. Although there is an abundance of data, it is distributed between different organizations who describe disparate parts of attacker behavior and network vulnerabilities. The challenge for the defensive operator is to turn distributed data into specific, coherent, and actionable knowledge.

MITRE publishes a widely-utilized framework based on the MITRE ATT&CK matrix [35], which starts by enumerating abstract attacker goals, and then drills down into how different vulnerabilities meet those goals, and eventually moves to the level of naming specific applications which are targeted by those vulnerabilities. Although these data make reference to one another on various MITRE and NIST websites, there is no comprehensive single source for the combination of this data from abstract goal down to the level of application.

Current automated cyber solutions for *active cyber security* typically focus on approaches like anomaly detection [2]. A complementary helpful utility for many cyber analysts, who work in a variety of areas such as hospitals, would direct sensor placement to single out known and unknown attack patterns or, patterns from attacks

that have recently increased in frequency recently, like ransomware did in 2019 [4].

1.1 Research Questions

In this thesis, we present two systems: **BRON** and **CHUCK** (Cyber Hunting Using Common Knowledge). **BRON**. **BRON** is an organized collection of public threat data that addresses the issue of unorganized and unevaluated public threat data and **CHUCK** is a tool that combines co-evolutionary algorithms with **BRON** to identify ideal sensor placement. We would like to answer the following research questions using **BRON**:

1. (a) Does the structure BRONdb allow users to easily find pieces of connected threat data?

   (b) How comprehensive is publicly available threat data?

   (c) How can we connect BRONdb to existing networks use it to find which pieces of threat data affect particular parts of the network?

We would like to answer the following research questions using **CHUCK**:

2. (a) How can we utilize competitive co-evolutionary algorithms as part of an active cyber security system?

   (b) How can we combine competitive co-evolutionary algorithms with existing public threat data?

   (c) A number of coevolutionary algorithm variants can serve to model the escalating arms race of attack and defense counter-measures, see e.g. [29, 8]. We evaluate several new and existing co-evolutionary algorithms and want to determine which co-evolutionary algorithm performs best in an active cyber security system?

1.2 Contributions

A key part of **BRON**, is BRONdb, a relational schema that links together threat information from abstract attacker goals all the way down to specific affected applications, as well as a set of analysis tools for analyzing the vulnerabilities present on any input network. **BRON** combines publicly available threat data into an easily accessible format. The contributions of **BRON** are:

- A relational schema that stores public threat data with varying levels of detail
- Analysis tools that evaluate the quality and comprehensiveness of the public threat data
- A tool that identifies the most damaging attack patterns on a given network

BRON, which translates to 'the bridge' in Swedish, can be thought of as a 'bridge' between the many different sources of publicly available threat data. In **CHUCK**, we utilize **BRON** ('cross the bridge') as part of an *active cyber security* utility that pinpoints sensor placements. **CHUCK** is initialized with a description of the network that needs to be protected. **CHUCK** exploits competitive co-evolutionary algorithms to identify a network's optimal active sensor placement and patch modifications in several different attacker scenarios. The contributions of **CHUCK** are:

- We present a utility, **CHUCK**, that bridges a gap between the retrospective, reactionary stance of current network security and the need to anticipate hard-to-detect, high stealth, unobserved attack staging of known and unknown threats.
- We introduce a new coevolutionary algorithm variant named RECENT_COEV as described in Section 4.1.2
- We present a new application of a coevolutionary algorithm variant named LS_COEV in Section 4.1.2
- We evaluate, for the first time, to our best knowledge, evolved solutions on a network for which they were not evolved in Section. This "out-of-sample"

testing offers a machine learning oriented measurement and indicates the level of transferability a coevolutionary algorithm can support.

- We compare different coevolutionary algorithm variants, with out of sample testing
- We demonstrate how public threat data and automatic adaptation, via evolutionary algorithms, can support *active cyber security* in terms of sensing target identification.

The combination of **CHUCK** and **BRON** provide a useful utility for organizations like hospitals that have devices running legacy software (and cannot easily be upgraded).

1.3 Thesis Structure

In Chapter 2, we discuss the related works of **CHUCK** and **BRON**. In Chapter 3, we discuss unifying public threat data **BRON** and in Chapter 4, we discuss using **CHUCK** for ideal sensor placement. Lastly, in Chapter 5, we conclude and list several areas for future work.

Chapter 2

Related Works

In Section 2.1, we discuss different approaches for storing threat information. In Section 2.2, we describe different models and software available for risk assessment. In Section 2.3, we describe how we incorporate MITRE Attack data into **BRON** and **CHUCK**. In Section 2.4, we list different methods for active sensor placement. Lastly, in Section 2.5, we discuss co-evolutionary algorithms.

2.1 Frameworks for storing threat information

There have been several proposed frameworks in recent years that all work towards the common goal of succinctly describing, storing, searching, and propagating threat information. There are a huge number of frameworks and formats under which threat information is distributed, including but not limited to MISP [41], CTI [16], OVM [42], FireEye OpenIOC [7], STIX [28], IDS rules [32], OpenC2 [25], UCO [36], VERIS [40], and IODEF [3]. A very widely-used framework is published jointly by MITRE and the NIST, and it consists of a hierarchy of taxonomies that all relate to different techniques, tactics, and procedures that are part of a kill-chain. In this paper, we utilize the MITRE ATT&CK matrix and combine it with data from the NIST.

Table 2.1: Related open-source BAS products

Name	**Meta-Analysis**	**Recommendations**	**Taxonomy**
BRON/Chuck	Yes	Yes	ATT&CK/NIST
Uber Metta [39]	No	No	ATT&CK
CALDERA [20]	No	Maybe	ATT&CK
BRAWL [19]	No	No	ATT&CK
Atomic Red Team [31]	No	No	ATT&CK
Infection Monkey [10]	No	Yes	Ambiguous, likely ATT&CK
NeSSi [5]	No	Yes	Ad hoc

2.2 Risk Assessment Tools

There are many different ways to perform risk assessment of a network. For example a mathematical model was proposed by Janiszewski et al. in 2017 implements a risk calculation model within a real system in order to provide better tools for software management. Their model of risk also depends on scores from the NIST/MITRE framework, but they limit themselves to only looking at a few vendors and their operating systems, rather than many types of software. We found that many current risk assessment tools use perform what is called 'Breach and Attack Simulations' (BAS). There are many other commercially and open-source BAS software. Since many commercially available products do not share implementation details, we compare **BRON** only to other easily found open-source efforts. In Table 2.1, we list some of the most popular open-source efforts that have the common objective of using threat information to assess network vulnerability. With the exception of BRAWL, these major players in the BAS space are agent-based rather than graph-based, and can take significant configuration. Only **BRON** provides a meta-analysis of its own data, allowing the user to more clearly understand the limitations of the underlying data. Recommendations on how to mend network vulnerabilities are available from some products, including **BRON**. It is also clear from this analysis that the use of MITRE ATT&CK taxonomy is pervasive in the community.

2.3 MITE Attack Data

As mentioned above, we use MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) Matrix™ which is " is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations." [35] along with NIST as our source of public threat data. The Matrix[1], see Figure 2-1a, contains information for common platforms such as Windows, macOS, Linux, AWS, Office 365, and SaaS. Each column of the Matrix is a tactic in an attack kill chain. For each tactic, the Matrix has a variable number of cells. Each cell enumerates a technique or procedure. Some cells also enumerate an attack pattern, often referred to as a CAPEC [21] (a MITRE acronym expanding to "Common Attack Pattern Enumeration and Classification"). **CHUCK** draws upon these CAPECs as attack patterns it considers. **CHUCK** needs to know whether any application running on its network is vulnerable to a specific pattern and it needs to know the severity of that vulnerability. To retrieve the severity, **CHUCK** relies upon a data collection and retrieval system called **BRON** [13] that can map from a CAPEC and network description to an application, the node it runs on, and a vulnerability score which indicates the severity,. **BRON** draws upon the MITRE CWE™ [24] (Common Weakness Enumeration Specification) which is "a community-developed list of common software security weaknesses" described in a common, text-based, semi-structured language. It also uses another set of repositories which are the NIST National Vulnerability Database's CVE [23] (Common Vulnerabilities Enumeration) Slice and its CPE [22] Slice (Common Platform Enumeration). Applications in a CVE are specified in a CPE Common Platform Enumeration, which is a naming specification for software and applications. **CHUCK** provides **BRON** with its network information specified as CPEs. For common vulnerabilities, the NVD CVE slice provides a Common Vulnerability Score [15] (CVS). This CVS has been assigned by domain experts according to a free and open industry standard for assessing vulnerabilities depending on the accessibility, integrity, and confidentiality of the application. **BRON** provides

[1]Per MITRE practice, we capitalize elements of ATT&CK Matrix™.

CHUCK this score which is in the range of 0 to 10.

ATT&CK Matrix for Enterprise

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Commonly Used Port	Automated Exfiltration	Account Access Removal
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Communication Through Removable Media	Data Compressed	Data Destruction
External Remote Services	Command-Line Interface	Account Manipulation	AppCert DLLs	BITS Jobs	Brute Force	Browser Bookmark Discovery	Component Object Model and Distributed COM	Clipboard Data	Connection Proxy	Data Encrypted	Data Encrypted for Impact
Hardware Additions	Compiled HTML File	AppCert DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	Domain Trust Discovery	Exploitation of Remote Services	Data from Information Repositories	Custom Command and Control Protocol	Data Transfer Size Limits	Defacement
Replication Through Removable Media	Component Object Model and Distributed COM	AppInit DLLs	Application Shimming	Clear Command History	Credentials from Web Browsers	File and Directory Discovery	Internal Spearphishing	Data from Local System	Custom Cryptographic Protocol	Exfiltration Over Alternative Protocol	Disk Content Wipe
Spearphishing Attachment	Control Panel Items	Application Shimming	Bypass User Account Control	CMSTP	Credentials in Files	Network Service Scanning	Logon Scripts	Data from Network Shared Drive	Data Encoding	Exfiltration Over Command and Control Channel	Disk Structure Wipe

(a) Screen shot of the MITRE ATT&CK matrix. Columns are tactics and cells are techniques

2.4 Active Sensor Placement

Multiple security software tools, **BRON** one among them, rely upon one or more of ATT&CK, CWE, CVE and CPE [1, 9]. To understand an attack's impact, they set up an attack graph which is a representation of all paths through a system that end in a state where an adversary successfully achieves his goal [27]. In contrast, **CHUCK** uses an easily derivable network description, in CPE format. One prior work identifies critical attack assets in dependency attack graphs, e.g. [34]. Others study topological vulnerability analysis and analyze vulnerability dependencies to show all possible attack paths into a network. See [12] for this being done on the CVE level. Work on optimal sensor placement for intrusion detection and alert prioritization also relates to **CHUCK**. Again, it draws upon attack graphs, e.g. [26]. The sensor-placement problem used in [26] is an instance of the NP-hard minimum set cover problem and it is solved with an efficient greedy algorithm that depends on an attack graph. All of these studies assume static adversaries in contrast to how **CHUCK** assumes adaptive adversaries and uses an more easily derived network description.

2.5 Using Coevolutionary Algorithms

Coevolutionary algorithms explore domains in which the quality of a candidate *solution* is its ability to pass a set of *tests*. Conversely, a *test*'s quality is its ability to fail (not pass) a set of *solution*s. In competitive coevolution, similar to game theory, the search can lead to an arms race between *test* and *solution*, both evolving while pursuing opposite objectives [29].

A coevolutionary algorithm evolves two populations with selection and variation using standard selection, crossover and mutation techniques. One population comprises attackers and the other defenders. In each generation, engagements are formed by pairing an attacker and a defender. Each attacker–defender pair in the engagement environment is assigned a score. Fitness is then calculated over all an adversary's engagement scores. The populations are often evolved in alternating steps: first the attacker population is selected, varied, updated and evaluated against the defenders, and then the same for the defender population. We name this standard coevolutionary algorithm Coev.

CHUCK uses Grammatical Evolution (GE) which is a type of evolutionary algorithm. GE has been used in several adversarial domains in cybersecurity [11, 33, 8]. It uses a Backus Naur Form (BNF) context-free grammar and an intermediate interpreter to map from the "genome" to a "phenome" that expresses an executable behavior. Like all EAs, variation occurs on the genome (in GE, an integer sequence) and fitness depends on the phenome. In GE the interpretation step raises locality issues, however the grammar and the rewriting assure syntactically valid offspring [38]. A grammar allows the representation of candidate solution behavior to be easily customized and expressed in direct domain vocabulary. Grammars also offer design flexibility: changing out a grammar and the environment of behavioral execution does not require any changes to the rest of the algorithm.

Chapter 3

Unifying Threat Data-BRON

BRON[1] contains two parts: BRONdb and analysis tools. BRONdb is a relational schema that serves as a central location for the currently unorganized and disperse public threat data. The analysis tools in **BRON** provide methods for evaluating the quality of the public threat data, which is not currently done. We answer research questions 1a, 1b, and 1c in this chapter.

3.1 Methods

An overview of MITRE and NIST threat data types are described in Section 3.1.1. BRONdb is discussed in Section 3.1.2. In Section 3.1.3, we discuss network representation. Network analysis and scoring is discussed in Section 3.1.4.

3.1.1 Data sources and Acronyms

Six different threat data types are collated in **BRON**. A summary of each data type is described in Table 3.1, and we give a succinct description below. We used the 2019 version of these data types in building **BRON**.

Tactics and Techniques The tactics and techniques come from MITRE's Enterprise ATT&CK Matrix [18]. These represent the most general, abstract attacker

[1]This chapter of the thesis was done in collaboration with Bryn Reinstadler.

Table 3.1: Meaning of acronyms for the data sources

Acronym	**Full Name**	**Organization**	**Data type**
TACTIC	Adversarial Tactics	MITRE	Documents common tactics that advance persistent threats used against networks
TECHNIQ	Adversarial Techniques	MITRE	Documents common techniques that advance persistent threats used against networks
CAPEC	Common Attack Pattern Enumeration and Classification	MITRE	Catalog of attack patterns with a comprehensive schema and classification taxonomy
CWE	Common Weakness Enumeration	MITRE	Nomenclature and dictionary of security-related flaws in architecture, design, or code
CVE	Common Vulnerabilities and Exposures	NIST	Nomenclature and dictionary of security-related flaws in software and applications
CPE	Common Platform Enumeration	NIST	Naming specification for software and applications

strategies. Tactics consist of 12 high-level goals that are part of the kill-chain, such as *Initial Access*, *Persistence*, or *Exfiltration*. Each of the 12 tactics references at least one technique. Techniques tend to be focused around lower-level details about how the adversary actually executes its different goals.

Common Attack Pattern and Enumeration (CAPEC) Techniques make reference to Common Attack Pattern and Enumeration (CAPECs) [21], which list the most common types of attack patterns.

Common Weakness Enumeration (CWE) A CWE [24] describes the different types of weaknesses that occur in software; these are linked to by CAPECs above. The weaknesses are not tied to any specific product or software.

Common Vulnerabilities and Exposures (CVE) CWEs list Common Vulnerabilities and Exposures (CVEs) [23], which are known vulnerabilities of major software applications or operating systems. Each individual CVE has a numerical score that is part of the Common Vulnerability Scoring System (CVSS) [15]. The higher the score, the more damaging a CVE is to a network. CVSS scores have a maximum of 10. **BRON** uses this score to determine the riskiest applications and nodes on a network. Currently, **BRON** contains the 2019 CVE data and future work will incorporate earlier CVE data.

Common Platform Enumeration (CPE) To connect the vulnerabilities and weakness to applications and operating systems, we use the Common Platform Enumeration (CPE) [22]. A CPE [22] is a standardized way to specify application and operating systems. We will refer to applications and operating systems that are specified in the CPE format as App-Platform.

3.1.2 BRONdb

Implementation overview BRONdb is the schema which contains the different types of threat data and their connections to other pieces of threat data. BRONdb

represents threat data pieces as nodes in a graph, and connects nodes with an edge if two pieces of threat information are connected. The nodes carry information on the specific piece of data, and their form has been standardized to make use and extension as easy as possible. A simplified schematic of the BRONdb is given in Figure 3-1.

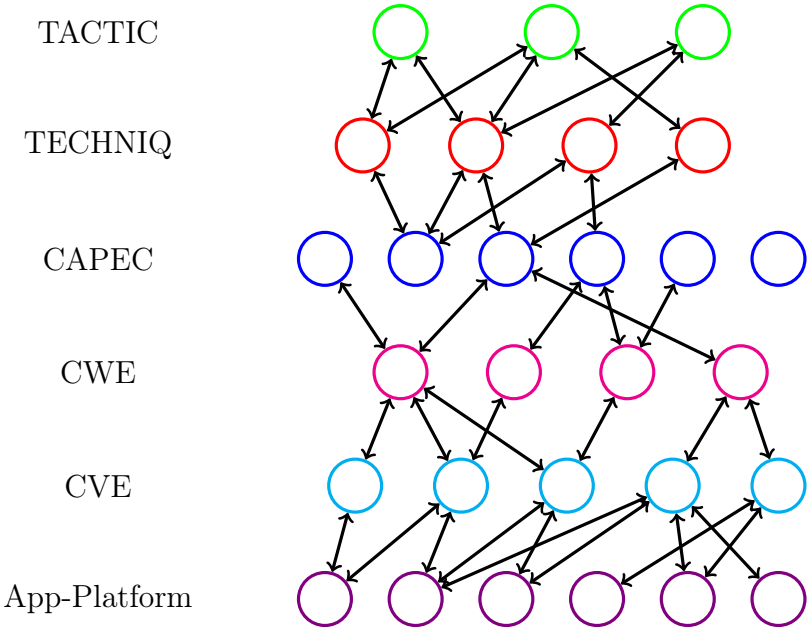

Figure 3-1: A simplified schematic of BRONdb, showing the connected graph of the data. Note that some nodes are not connected to any other data layers, and some are only connected to the next or the previous data layer, but not both.

Searching BRONdb Since we store the threat data in a graph format, **BRON** has bidirectional searching capability all the way from abstract attacker goals such as *Persistance* all the way down to the specific applications that can be targeted as part of an attack. Users of **BRON** may query with any piece of threat data (e.g., A CAPEC, or a set of CWEs) and ask for any type of threat data (e.g., a set of TACTICs or a list of App-Platform).

For example, consider a hospital that is faced with an attacker who wants to persist in the network (using the persistence tactic). Given the tactic of persistence, **BRON** produces all the possible paths from techniques, CAPECs, CWEs, CVEs, and CPEs that start at persistence (see Figure 3-2). **BRON** shows that the tactic persistence

is connected to 63 techniques, 18 CAPECs, 16 CWEs, 739 CWEs, and 1,352 App-Platform. A hospital could utilize the information shown in Figure 3-2 in several different ways. The hospital can also use **BRON** to trace individual data paths as well. For example, it could follow an attacker path that uses a persistence tactic that has an accessibility technique and uses an attack pattern of Replaced Trusted Executable. This attack pattern exploits a weakness of Improper Access Control which has 401 linked vulnerabilities with a total CVSS score of 6,106. The information from **BRON** shows that there are many vulnerabilities that link to Google Chrome. To help mitigate the risk, the hospital could perform appropriate patches for Google Chrome or switch to a more secure browser. However, it should be noted that the more frequently a product is used and tested, more vulnerabilities are reported. Future work should report vulnerabilities based on market share so this is taken into account. The hospital can use such information to determine what attacks they are most susceptible too.

3.1.3 Network Representation

In addition to providing a relational schema of the different threat data, **BRON** also contributes the capability to find vulnerable areas in actual network configurations. In this work, we present two example network configurations and give an analysis in Section 4.2.2.

BRON takes as input networks which are represented by a flat listing of four types of entities: servers, clients, routers, and firewalls. We refer to the collection of entities as *network nodes*. Some of the nodes are connected to each other, while others exist independently within the network.

Each network node contains a listing of applications. The App-Platform are listed in CPE format which allows the network configuration to be analyzed using data in BRONdb. Specifically, if a CVE contains a App-Platform and that App-Platform is running on at least one node in the network, then **BRON** considers that CVE to affect the network node containing the App-Platform and the overall network. The pre-specified networks that we use are provided by domain experts.

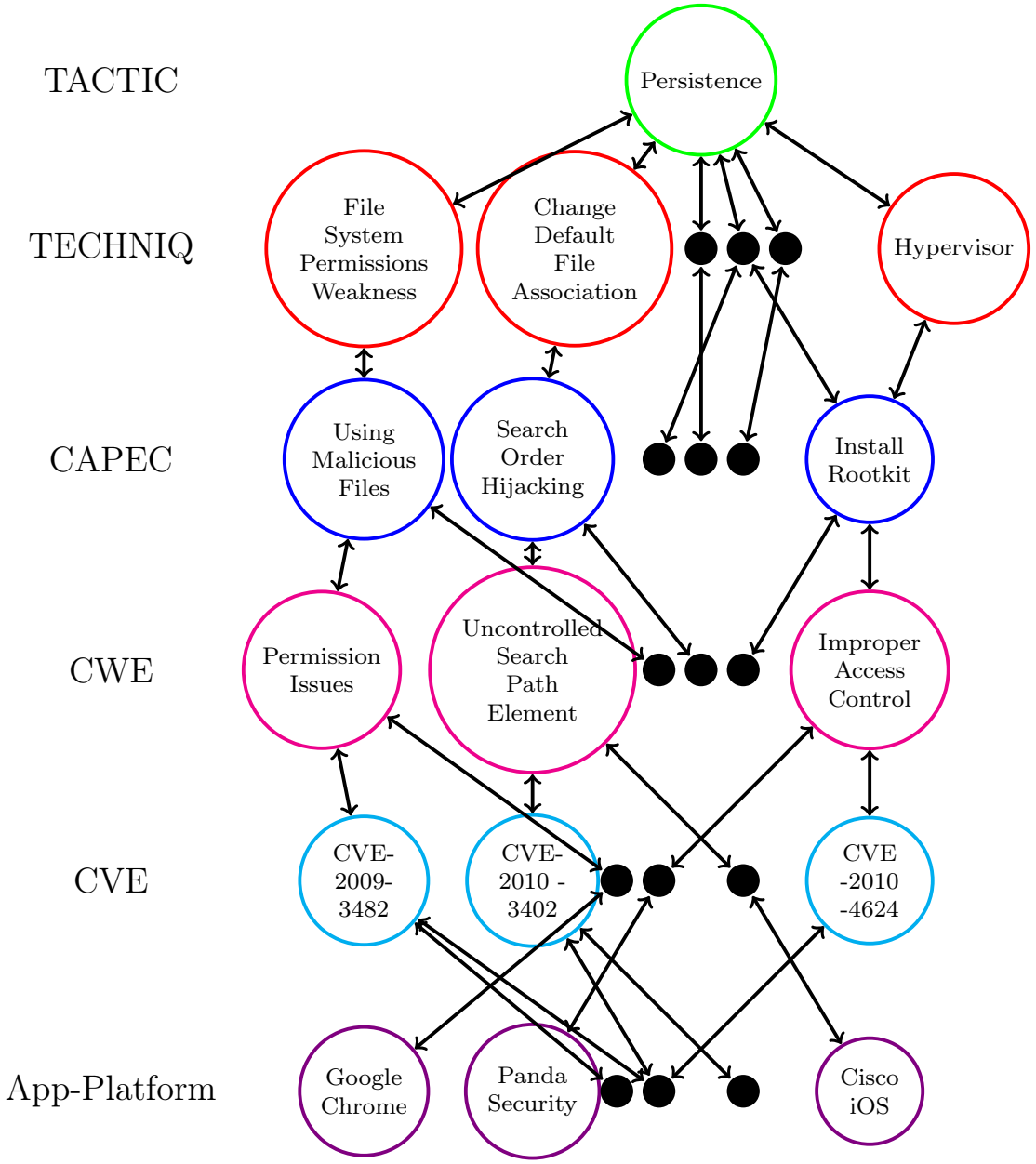

Figure 3-2: Overview of paths provided by **BRON** for an input of tactic persistence. Given a single tactic, **BRON** shows all of the attached techniques, all of the CAPECS attached to the techniques, all of the CWEs attached to the CAPECs, all of the CVEs attached to the CWEs and all of the App-Platform attached to the CVEs.

3.1.4 Network-Specific Analysis

Running a network-specific analysis consists of two steps. First, we construct a network-specific BRONdb which connects nodes in the network to the App-Platform in the full BRONdb, see Figure 3-3. Then, we can calculate risk scores or do other analyses by traversing the network-specific graph.

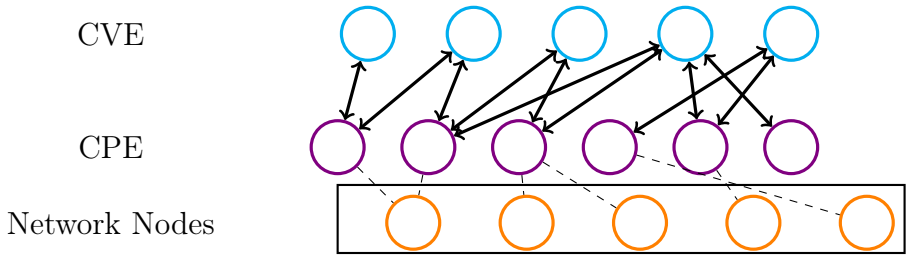

Figure 3-3: A simple schematic showing how we can connect BRONdb to a network by creating edges between network nodes, which have App-Platform listed, and the App-Platform-containing nodes in BRONdb. The upper layers of the BRONdb (higher than CVE) are abbreviated.

Calculating a network-specific risk score The network-specific risk score is computed by iterating through all unique CVEs that are reachable from a network node and summing together the CVSS scores. See Algorithm 1 for the pseudocode.

Algorithm 1 Algorithm for computing total risk score

```
total_risk_score = 0
unique_cves = []
for node in network nodes do:
    for cpe in node do:
        for cve in cpe: do
            if cve not in unique_cves then:
                total_risk_score += cve_risk_score
                unique_cves.append(cve)
```

Determining the riskiest CPE Analogous to the above, we determine the riskiest App-Platform by iterating through all CVEs reachable from App-Platform in the network and taking the per-App-Platform sum of the CVSS scores. The App-Platform with the highest score is the riskiest App-Platform. Note that we do not have to keep

track of unique CVEs because there are no repeated edges from CVE nodes to App-Platform.

Algorithm 2 Algorithm for computing riskiest App-Platform

```
max_score = -1
riskiest_App-Platform = None
for App-Platform in app-platforms do:
    if App-Platform in network_nodes then:
        App-Platform _score = 0
        for  cve in App-Platform do:
            App-Platform _score += risk score for cve
        if App-Platform _score > max_score: then
            riskiest_App-Platform = App-Platform
```

3.1.5 Searching on a network-specific BRONdb

Bidirectional search is also supported on a network-specific BRONdb, in a way analogous to search on the full BRONdb. However, in this case, search can also start from or end at a node in the network.

Network specific analysis could be quite useful for a hospital. Consider a hospital where several X-Ray machines are running legacy software [37] and the cyber analysts at the hospital would like to determine the risk the old software poses. A hospital could create a network-specific BRONdb and determine several useful data points such as the attacker patterns and vulnerabilities that are a result of the X-Ray machines, the risk that other nodes in the network have because of the X-Ray machines and the risk the X-Ray machines pose compared to overall risk to the network. These pieces of information can help guide a hospital on what steps to take about the X-Ray machines.

3.2 Experiments

The goals of our experimental section, were to analyze the comprehensiveness of publicly available threat data as well as determine how we can use a network-specific BRONdb to gather risk information about a network. To answer these questions, we

perform a meta-analysis on the data within BRONdb, and we give an example of how BRONdb can be used to analyze two different network topologies provided by a domain expert.

3.2.1 A meta-analysis of BRONdb

First, we wanted to do a simple meta-analysis on the contents of BRONdb. We examined all of the threat data and connections (nodes and edges) in BRONdb, and show here the connectivity between the various layers in graphical format (Figure 3-4). Our goals for the meta-analysis were to evaluate the quality of the public threat data as well as the quality of the risk assessment that **BRON** provides.

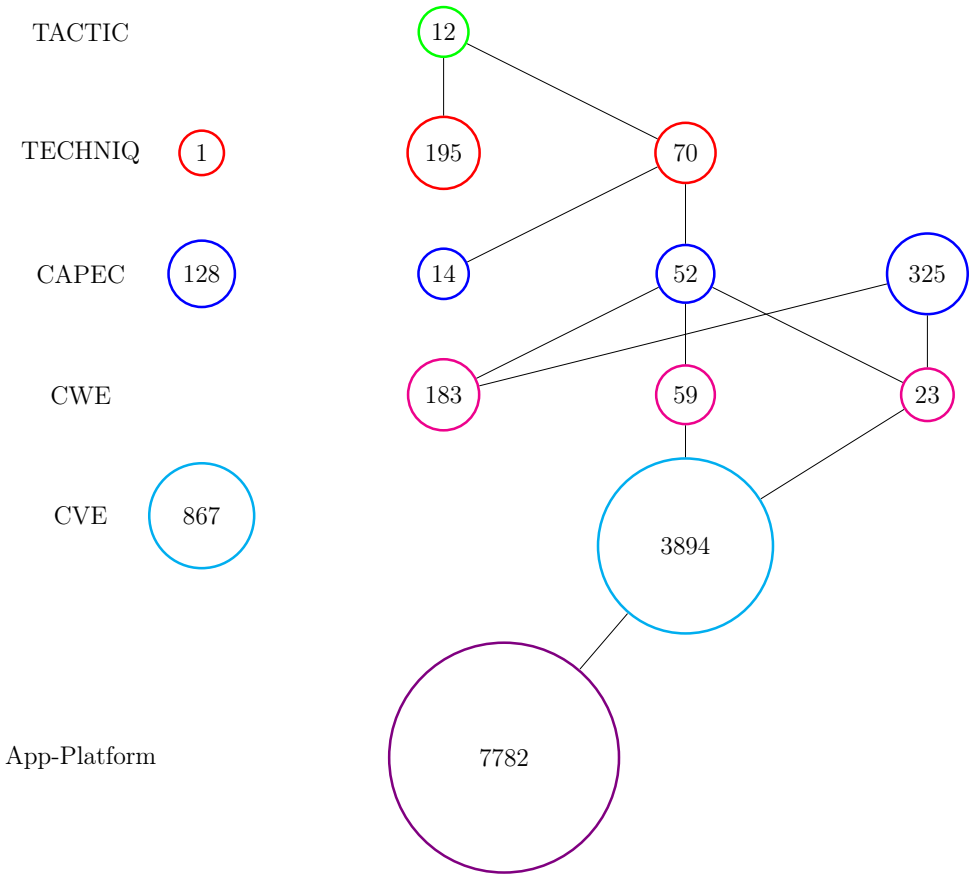

Figure 3-4: Visualization of the contents of BRONdb. Note that the number of nodes at each threat level that are unconnected. In particular, there are many CVE nodes that are not connected to a single App-Platform.

It is also clear that the connections between threat data in the MITRE and NIST

data are not a straightforward path from abstract goals (TACTICs) down to afflicted applications (App-Platform). Instead, in most layers, there are some orphan nodes which are not connected to other data types, there are nodes that are only connected to 'parent' data types, some nodes that are only connected to 'child' data types, and other nodes that are connected to both 'parents' and 'children'. This result highlights gaps in the currently available threat data which are not being accounted for by other threat assessment technologies on the market. It also gives guidance for domain experts who may be willing to link these data together to ensure a more comprehensive MITRE/NIST database.

Because of the large number of orphan nodes, especially in CVEs, we were curious about the amount of 'floating' risk; that is, risk scores that have been given to CVEs which are not yet annotated with any App-Platform, such that they cannot be easily be linked to a network. We found that 18.2% of all CVEs (867 / 4761) were not linked to any App-Platform, and that these CVEs accounted for 19.7% of the total risk (sum of the risk scores of all the CVEs). Over 55% of the most risky CVEs, those with a risk score ≥ 9, are orphan nodes. This data shows that on average the floating CVEs are rated to be more risky than the CVEs which are attached to App-Platform, and therefore which are attachable to networks. The distribution of risk is shown in Figure 3-5. Therefore, through this meta-analysis, we have identified a high-impact area where more work can be done to connect high-risk CVEs to App-Platform to enable better defense.

3.2.2 Health Related Data Meta-Analysis

Performing a meta-analysis on available data can also be of use to a hospital as well. A hospital could look at a group of medical related CVEs and determine how much of a threat the CVEs are to the hospital. For example, we used BRONdb to look at the paths of 9 2019 CVEs that are on the CVE website list that contain the word 'health' in the description. A table with the CVE IDs and descriptions can be seen in Table 3.2. 3 of the CVEs were not found in BRONdb, indicating a discrepancy between the website and the downloads list. Each of the remaining 6 CVEs connects

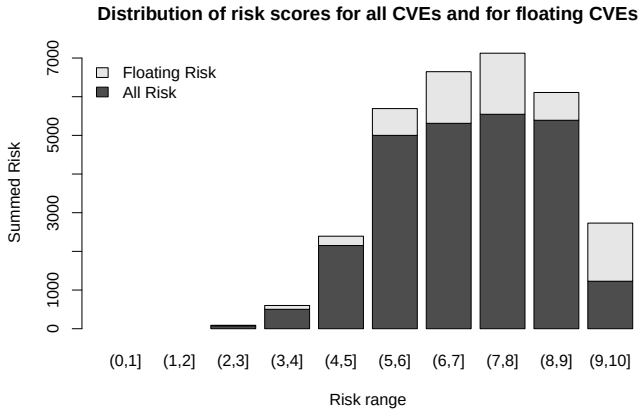

Figure 3-5: The distribution of the risk scores overall, highlighting the floating risk which is not connected to any App-Platform.

CVE-ID	Summary
CVE-2019-4546	Additional privileges that are not allowed for some users in IBM Maximo Health- Safety and Environment Manager
CVE-2019-2629	Vulnerability in Oracle Health Sciences Data Management Workbench
CVE-2019-2432	Vulnerability in the Oracle Argus Safety (Login)
CVE-2019-2431	Vulnerability in the Oracle Argus Safety (Console) and requires two users
CVE-2019-2430	Vulnerability in the Oracle Argus Safety (Console) and requires one user
CVE-2019-17390	Privilege escalation in Health Monitor Service
CVE-2019-15563	SQL Injection in Observational Health Data Sciences and Informatics
CVE-2019-11231	Issue with GetSimple CMS
CVE-2019-10686	An SSRF vulnerability was found in an API from Ctrip Apollo

Table 3.2: Names of 9 2019 Health related CVEs

to no more than 2 App-Platform and 1 CWE. The range of the CVSS scores are 5.2-9.8. The CVE with a CVSS score of 5.25 connects to most number of different CAPECs (57) while some of the other CVEs connect to 0 CAPECs. **BRON** allows us to look at the connections between the different data types as seen in Figure 3-6. We see some 'information gaps' in the data of the 9 CVEs. Three of the CVEs are 'floating' and most of the CAPECs are not connected to a single technique. While more information is needed to get the full threat picture of these CVEs, a hospital can still use the CVEs to make informed security decisions.

3.2.3 Finding Risk in Networks using BRONdb

We were also interested in having the ability to assess specific network topologies. Here we showcase some analysis strategies for assessing threats on two different custom network set-ups; we note that any user may create their own network as explained

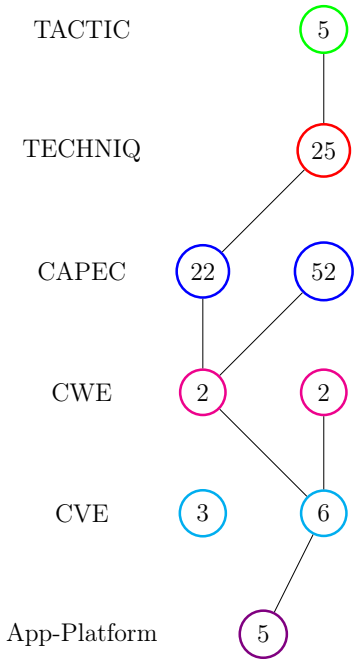

Figure 3-6: Visualization of the connections for the 9 health CVEs. Note that in many of the threat data types we see nodes that are only connected to layer below. The 5 tactics are privilege escalation, persistence, lateral-movement, defense-evasion and discovery.

above and may run their analyses using the tools already set up in our public domain repository. A visualization of the large and small network we analyze can be seen in Figure 3-7 and Figure 3-8 (respectively). Each node represents a device that has an IP address and a role. Roles are different servers, routers, and clients. Subnets are made up of devices that share an IP address. We began our network-specific analysis by looking for vulnerabilities in two networks, one small and one large, both built by domain experts; the results are shown in Table 3.3. First, we summed the total risk over the network and then reported that risk as a percent of the total, non-floating BRONdb risk (the highest possible risk score for any network). Notably, it is unlikely that a network will reach 100%, since many App-Platform are different versions of the same software.

Our analyses can suggest the riskiest software currently installed in a network, as well as identifying the riskiest node, as calculated by the cumulative unique CVE

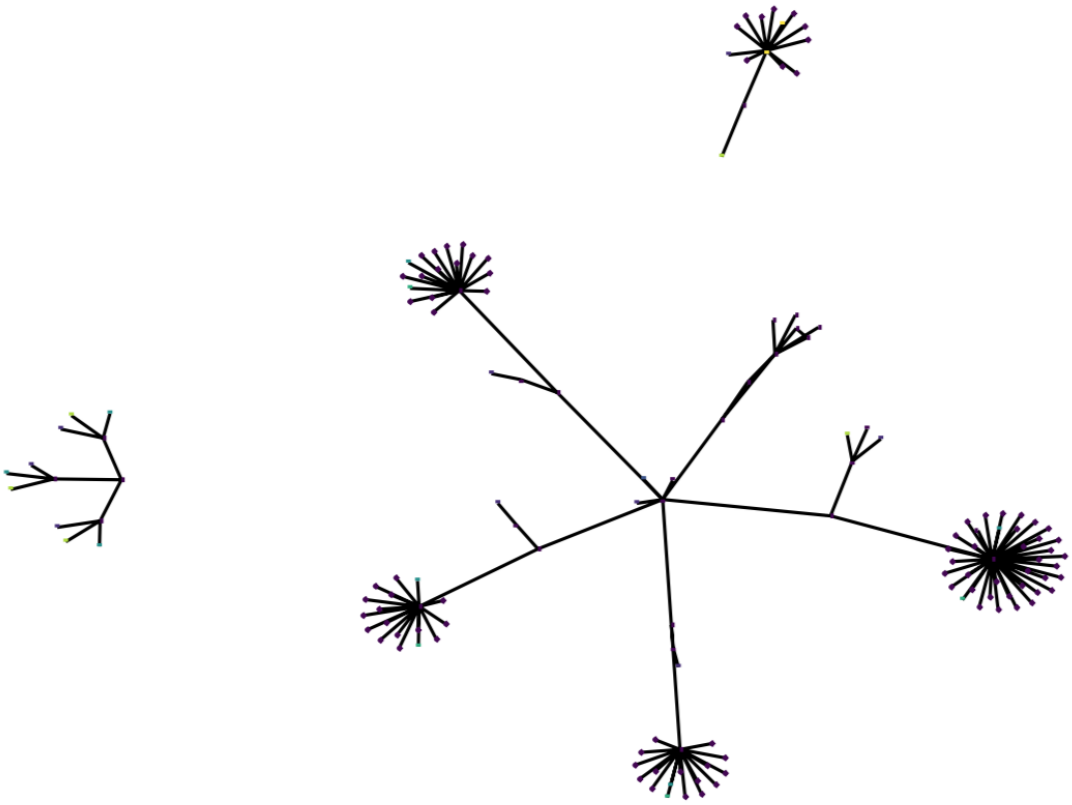

Figure 3-7: Visualization of the large sample network configuration. There are 203 subnets, 146 edges and 147 nodes. The different colors represent different roles.

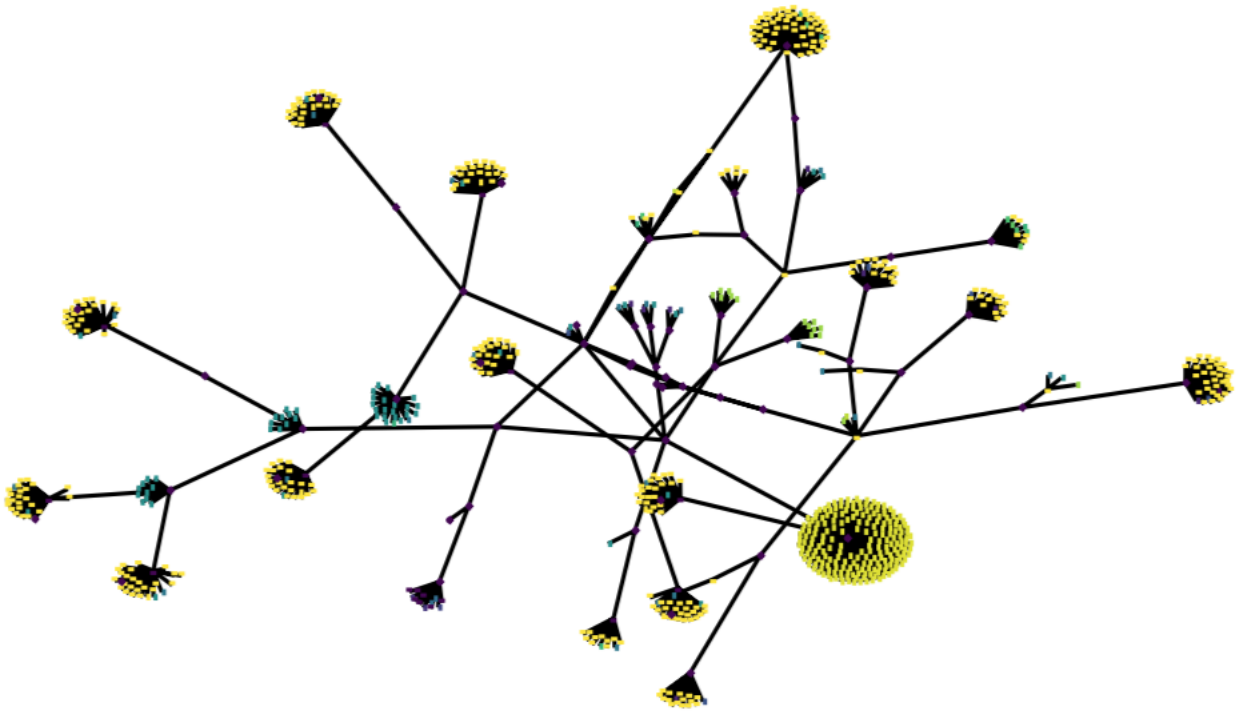

Figure 3-8: Visualization of the small sample network configuration. There are 927 subnets, 789 edges and 787 nodes. The different colors represent different roles.

score sum. Some risky software may not be avoidable (such as Mac OS Excel 2019), but others may be; it is a useful tool for network administrators to see which of their currently-installed software packages are most likely to come with vulnerabilities, as this may influence them to update or upgrade these vulnerable softwares on a regular schedule. In addition, better sensor placement may be possible when cyber defenders are aware of the most vulnerable pieces of a network. Sensors may, for example, be placed at the riskiest nodes found by **BRON**.

BRON is a useful tool that bridges together and evaluates many different pieces of public threat data. A key part of active cyber hunting, is using such data to identify vulnerable nodes in a network. As described in the next chapter, we 'cross the bridge' with **CHUCK**.

Table 3.3: Results of network-specific analyses

	Sum of unique, non-floating CVEs	Score (%)	Riskiest Software	Riskiest node
Small network	2126.6	8.4	e.g. Mac OS Excel 2019	e.g. DODINF1
Large network	2204.9	8.7	e.g. Windows Server 2008	e.g. A@A1RngInt04
Total BRON	25206.8	100	e.g. Junos 16.1	NA

Chapter 4

Determining Ideal Sensor Placement-CHUCK

Within any given network, there will always be nodes that are more vulnerable to attack than others. As part of active cyber hunting, cyber analysts would like to find the most vulnerable nodes and place sensors on them. There is no definitive way to determine which node is the most susceptible to attacks. One could use **BRON** to find which nodes have the highest risk according to public threat data. However, relying just on recorded data does not take into unknown attackers. For example, if an attacker used a combination of patterns to hack into a CT machine, then the hospital's security system would not detect the attacker if only currently known attackers are considered. **CHUCK** brings both of these ideas together by using co-evolutionary algorithms and public threat data to identify nodes for optimal sensor placement. We answer research questions 2a, 2b, and 2c in this chapter.

4.1 Methods

We present the threat model in Section 4.1.1. In Section 4.1.2 we present descriptions of the parts of **CHUCK**.

4.1.1 Threat Model

Our threat model takes the perspectives of both an attacker and a defender. It is shown in Figure 4-1. Observed from left to right, it shows how an attacker adopts a threat and chooses a number of tactics to accomplish it. A tactic is then translated to one or more attack patterns, techniques or procedures, (often malware), that inform the attacker of what version of software on the network is vulnerable and could be maliciously targeted. The attacker only selects techniques for which the network runs their software versions While typically an attacker may not have complete network knowledge, **CHUCK** adopts this model to make the attacker as strong as possible, helping the defender to consider the worst case. **CHUCK** can optionally restrict the attacker's knowledge of the network. The defender, conceptually posing as an attacker, accesses the same information and "sees" the attacker's campaign allowing it to mitigate what they believe the attacker will optimally choose to do, with patches and active sensor placement, subject to budget limitations.

Both attacker and defender access and rely upon publicly available threat data (broken down into tactics, techniques, attack patterns and procedures) per Section **??**. While the attacker sets up campaigns of techniques, attack patterns or procedures, they are also able to exploit an application vulnerability that is not presently linked to publicly known techniques. Its objective is to maximize the severity of its attack. The defender targets software patches and active sensing locations on nodes running vulnerable software to minimize the severity of an attack.

4.1.2 CHUCK

Architecturally, **CHUCK** consists of two modules: a search engine that uses EAs or competitive coevolutionary algorithms and the BRON framework which provides the search engine with fitness scores. Figure 4-2 depicts the high level architecture of **CHUCK** and its evolutionary adaptation. **CHUCK** is initialized with a description of the network that is to be examined by both adversaries. It initializes a population of attack campaigns and a population of defender action-sequences that either patch

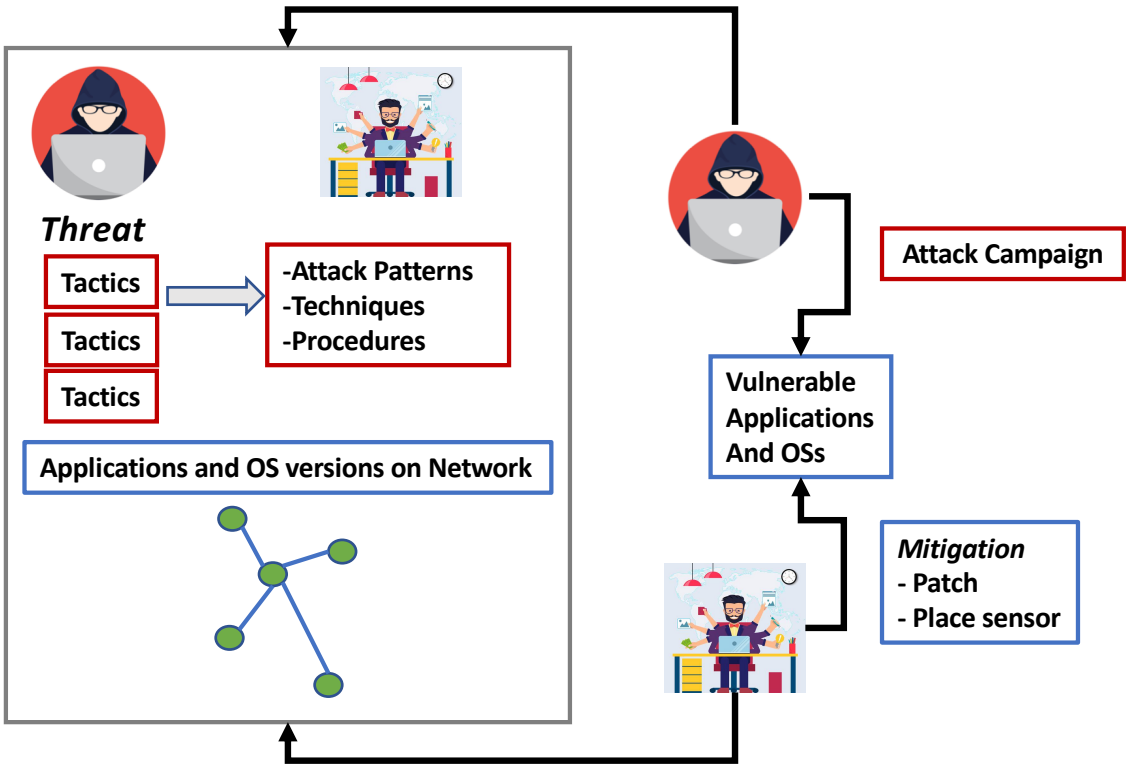

Figure 4-1: The **CHUCK** threat model. From left to right, we see how an attacker adopts a threat and chooses a number of tactics to accomplish it. The tactic is then translated to one or more attack patterns, techniques or procedures, (often malware), that inform the attacker of what version of software on the network is vulnerable and could be maliciously targeted. The defender, accesses the same information and "sees" the attacker's campaign allowing it to mitigate what they believe the attacker will optimally choose to do, with patches and active sensor placement, subject to budget limitations.

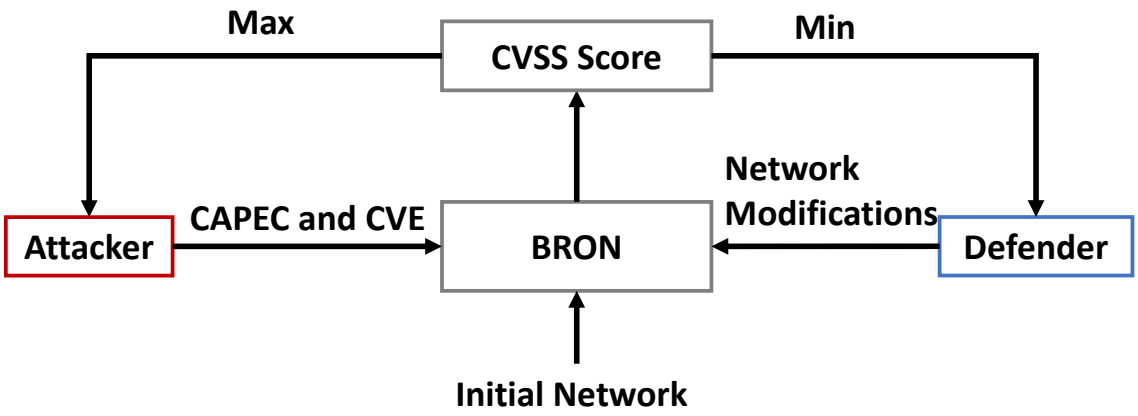

Figure 4-2: The **CHUCK** algorithm from a high level. See text for narrative.

software or place active sensors on specific nodes and applications. Each generation **CHUCK** draws every pairwise combination of attack campaign and defense action sequence from the two populations. For a pair, it first changes the network description to enhance the security via the patches and active sensing placements of the defender's action sequence. A patch changes the version of software which zeros the vulnerability severity and active sensing also zero's the vulnerability severity by effectively making the technique undesirable for the attacker. **CHUCK** then decomposes the attack campaign and passes its attack techniques, patterns and procedures, now simply called "patterns", one at a time to BRON along with the updated network description. BRON uses both units of information to identify whether there is application or operating system (OS) software running on the network that can be targeted by the pattern and reports the location and severity of the software vulnerability, if there is one. **CHUCK** sums these severities to assign a fitness to the attack, given the defense and assigns the reciprocal fitness to the defense. The final fitness of a population member is the sum of all engagement fitnesses. **CHUCK** then uses selection and variation to adapt each population before the next generation starts.

Adversary Grammars

We describe the campaign search space of the attacker with a context free grammar in Backus Naur Form (BNF), see Figure 4-3. CAPECs in public databases are iden-

Figure 4-3: Attack Grammar

```
<attack> ::= <threat>, <cve_values>
 <threat> ::= <capec><capec><capec>
 <capec> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ...
 <cve_values> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6| 7| 8| 9 ...
```

tified by a numerical tag. Our attack grammar allows these tags to be referenced. Additionally, the attack campaign contains a CVE value that does not have to be associated with a CAPEC. This allows the attacker to include a known CVE but one that the defender is unaware of, using a threat-based, tactical stance.

The defense grammar consists of a list of network mitigations. The defender chooses nodes that will be patched or actively sensed. There are two kinds of node mitigaionss: to the OS or application ones. These are distinguished in the grammar to support the fact that nodes have a single OS running multiple applications. The defender has to choose between updating an OS version or an application. If the defender chooses to update an application, then it has to choose between using a patch or adding a sensor. **CHUCK** integrates a budget constraint into defending. Conceptually, it assigns a unit cost to both kinds of mitigations – patches and sensing. It insists on some maximum number of sensing mitigations to model their frequently high cost and allows unlimited patches up to the budget limit. At an implementation level we currently allow a defender only 4 mitigations with at most 3 being active sensor placements.

The defender is limited on the number of sensors it can use by the sensor budget. Lastly, the node value indicates what node in the graph should be modified. The mitigationss are applied to the network during the fitness evaluation.

Algorithm Variants

We compare three variants of coevolutionary algorithms: Coev (described in Section 2.5), Recent_Coev and LS_Coev.

Figure 4-4: Defense Grammar

```
<defense> ::= <mitigations>
<mitigations> ::= [<mitigation>,...]
<mitigationn>::= <os> | <application>
<os> ::= 'os',<os_node>
<application> ::= <sensor_add> | <patch>
<sensor_add> ::='sensor add', <app _node>
<patch> ::= 'patch',<app_node>
<app_node> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6| 7| 8| 9 ...
<os_node> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6| 7| 8| 9 ...
```

RECENT_COEV RECENT_COEV proceeds with two evolving populations, attack campaigns, $\mathbf{A} = A_1, .., A_n$, and defense modifications, $\mathbf{D} = D_1, ..., D_n$, plus a single "recent" attack campaign, A_R and a single "recent" defense modification, D_R. In a generation, the attack campaigns are evolved then competed against D_R and the defense modifications are evolved and competed against A_R. Then the algorithm assigns A_R the best performing attacker A^* and D_R the best performing defense before executing the next generation. In this way, both populations are always evolved against the most recent effective adversary. A pictorial representation of this can be seen in Figure 4-5.

LS_COEV The motivation for lockstep coevolution is to model real-world dynamics observed where attackers can iteratively improve its attacks against a fixed defense, and then the defenders react and then the cycle repeats. The setup for lockstep coevolution is shown in Figure 4-6. One population is called the locked population, specified by the `locked_population` parameter, and the other is the non-locked population. For each generation, the locked population will evolve for a number of sub-generations against the fixed non-locked population, determined by the `locked_population_generations` parameter. There are `locked_population_elite_size` many elites kept during these sub-generations.

The non-locked population will then evolve based on the final state of the locked population for that step. For the next generation, the locked population will either re-

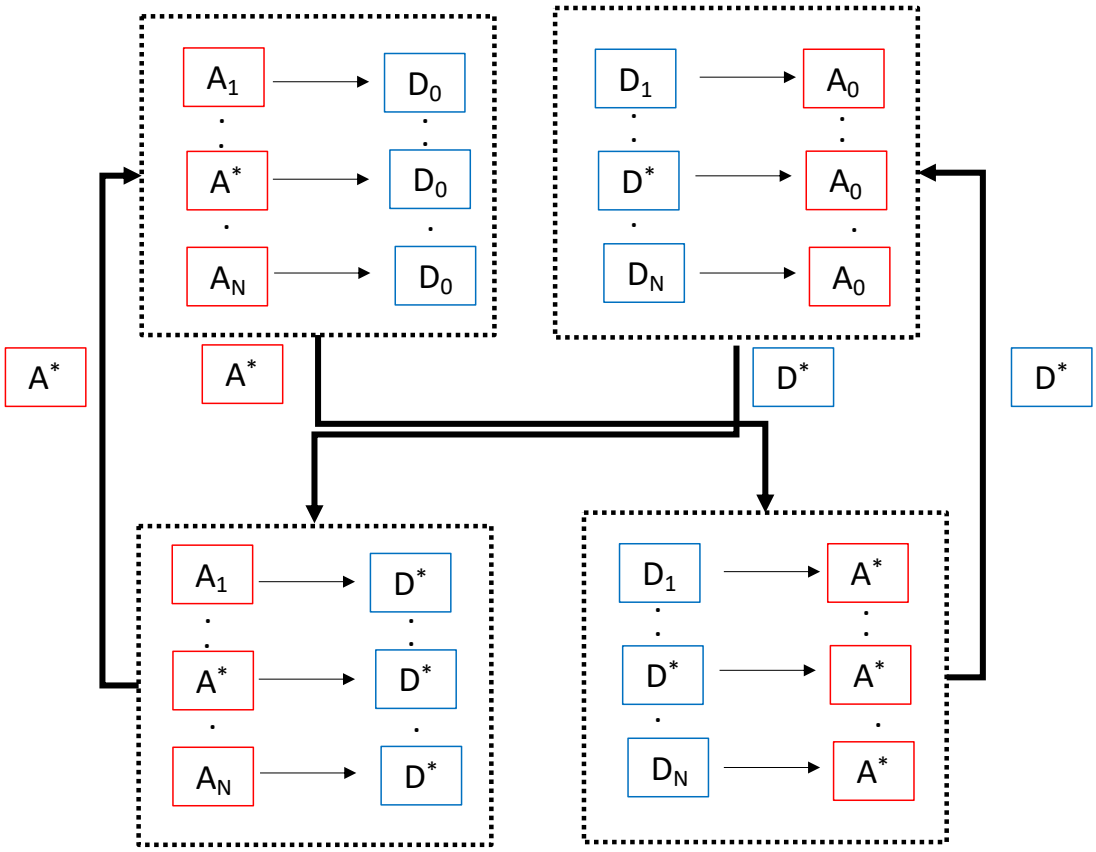

Figure 4-5: In RECENT_COEV, **CHUCK** uses the best performing attacker and best performing defender from the most recent round of evolution and competition to stand in as the sole adversary for the next generation.

set or evolve from using some number of parents, determined by a `locked_population_parents_size` parameter.

4.2 Experiments

In this section we experimentally demonstrate our implementation of **CHUCK** and perform an algorithms comparison. We answer our research question about which co-evolutionary algorithm variant has the highest performance. Section 4.2.1 provides experimental settings and implementation details. Section 4.2.2 describes experiments and their results.

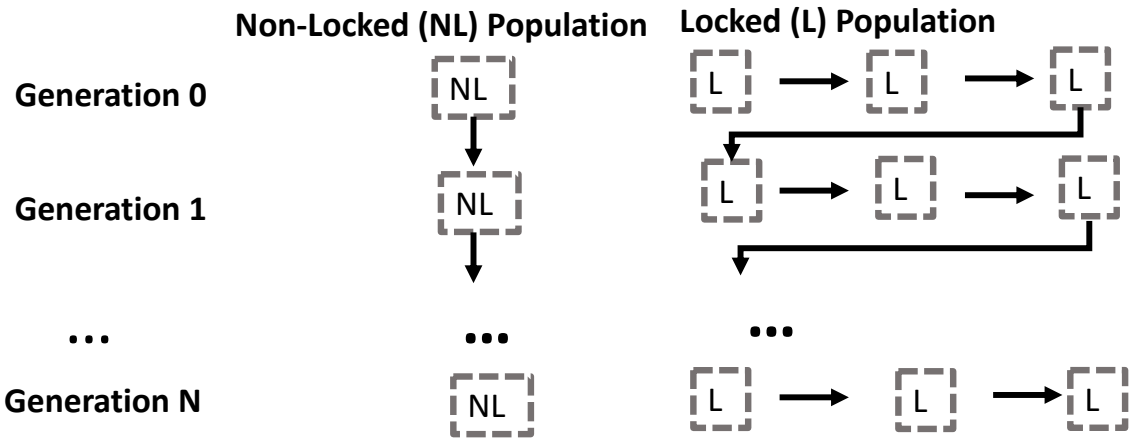

Figure 4-6: Lockstep coevolutionary algorithm.

Algorithm 3 LS_Coev

```
1: procedure PopulationStep(population, parents_size, adversaries)
2:     parents ← TournamentSelection(population, parents_size)
3:     new_individuals ← Variation(parents)
4:     for individual in new_individuals do
5:         individual.fitness ← AvgFitness(individual, adversaries)        ▷ MEU solution concept
6:     population ← GenerationalReplacement(new_individuals, population)
7: procedure LockstepCoevolution(populations, generations)
8:     t ← 0
9:     best_individuals ← ∅
10:    while t < generations do                                            ▷ run for # generations
11:        locked_pop ← populations_locked
12:        nonlocked_pop ← populations_nonlocked
13:        while t′ < locked_pop_generations do   ▷ locked population evolves against fixed adversary population
14:            PopulationStep(locked_pop, sizeof(locked_pop), nonlocked_pop)
15:            t′ ← t′ + 1
16:        PopulationStep(nonlocked_pop, sizeof(nonlocked_pop), locked_pop)
17:        PopulationStep(locked_pop, locked_population_parents_size, nonlocked_pop)
18:        best_individuals ← ExtractBest(populations)
19:        t ← t + 1
20:    return best_individuals                                             ▷ Returns best solutions found
```

4.2.1 Setup and Implementation

Network

We experiment with 2 networks provided by domain experts that are large and small in size. The large network has 787 nodes and 789 edges while the small network has 147 nodes and 147 edges. We group network nodes by type: servers, clients, routers, and firewalls. Each node also contains a list of the the applications that execute on it, specified in Common Platform Enumeration (CPE format) format. We store each flat listing as a json file. For each network, we build a corresponding adjacency graph

based on IP addresses and type. We separately store the connectivity information in this adjacency representation.

Fitness and BRON Inputs

Attacker campaigns use CAPEC formatting and defender mitigations use CPE formatting (plus a way to reference the JSON topology description). The inputs to BRON are the CAPEC attack patterns and network description after defender mitigations have been applied. BRON returns a value we term CVSS which is the CVS score. We then calculate fitness by summing the CVSS score over for all the attacker campaigns.

Experimental parameters

Experimental parameters can be found in Table 4.1 and our algorithm variants are named in Table 4.2. To identify the best attack campaign and/or defense mitigations from a run, for the large network, we run out of sample evaluation versus an unseen adversary. We select our unseen attackers and defenders manually from runs executed independently from these experiments. After a run, we select a pool of high performing individuals from the run, execute them on the adversaries they never saw during training, and designate the fittest one as the run's solution. The experiments are summarized in Table 4.2, each experiment is run on the large and small networks. The parameters we use for the evolutionary and co-evolutionary experiments are in Table 4.1. The next section describes the results from the experiments.

4.2.2 Results

We present the averaged results of 30 runs for each of the 5 algorithm variants, with both networks (large and small) in Table 4.3. The out-of-sample performance can be examined in Table 4.4. First, we will compare the overall results of the 5 algorithm variants and then we will look at each variant individually.

Table 4.1: Experiment parameters

Parameter	**Value**
Population size	20
Generations	20
Lock Step Population size	10
Lock Step Generations	2
Crossover Probability	0.9
Mutation Probability	0.1
Max Length	100
Tournament Size	2
Elite size	1
Number of runs	30
Attacker objective	maximize total CVSS
Defender objective	minimize total CVSS

Table 4.2: **CHUCK** Algorithm Variants

Evolve attackers vs a non-evolving defense.	`Attack`
Evolve defender on a non-evolving attack	`Defence`
Coevolve both populations, alternating	Coev
Coevolve with Lock-step	LS_Coev
Coevolve with Recency prioritization	Recent_Coev

Table 4.3: Average Best Fitness for the 5 algorithm variants and two network sizes. The best possible attacker fitness for the small network is 109411 and for the large network is 426814. The lowest defender fitness for the small network is -108224 and for the large network is $-427,690$

	Small Network		**Large Network**	
	Attacker Fitness	**Defender Fitness**	**Attacker Fitness**	**Defender Fitness**
Attack	109393 ± 0.0	N/A	426659.75 ± 0.0	N/A
Defense	N/A	-90566 ± 2260	N/A	-416691 ± 3302
Recent_Coev	108367 ± 0.0	-107835 ± 55	426813 ± 3	-419668 ± 50
Coev	105386 ± 1505	-104520 ± 1530	426648 ± 1564	-421485 ± 2445
LS_Coev	105782 ± 1313	-105031 ± 1450	426652 ± 1160	-423619 ± 1785

Table 4.4: Out of Sample Fitnesses

Variant	**Out of Sample Attacker**	**Out of Sample Defender**
Attack	N/A	Mean:426620, Min: 426526
Defense	Mean: -408782, Min: -426750	N/A
Recent_Coev	Mean:-408746,Min:-426714	Mean:426626, Min: 426526
Coev	Mean:-408623, Min:-426596	Mean:426617, Min: 426517
LS_Coev	Mean:-408591,Min:-426536	Mean:426620, Min: 426522

Algorithm Variant Comparison

To compare the performance across the 5 algorithm variants, we used out-of-sample testing. We report the average and minimum score as seen in Table 4.4.

We can first examine the results from the averaged 30 runs that we in Table 4.3 for a baseline comparison. In general, we would expect that a population that is evolving against a static population, would have a higher fitness value compared to two populations co-evolving with each other. Our results show this since the `Attack`and `Defence`variants had the highest performance.

As we can see from Table 4.4, the performance of the defender against the out of sample attacker improves as we try the more complex and realistic approaches to co-evolution. This indicates that the actual distribution of attackers is more reflective of attackers that evolve quite quickly compared to defense systems. We also see that the attackers found during Coev are not quite as strong as the attackers found during LS_Coev.

The results for the out-of-sample defender were not as definitive as the out-of-sample attacker results. There are a small number of CAPECs that link to a large number of CVE risk scores. These 'powerful' CAPECs were found in all of the strongest attackers from each of the evolution and co-evolution experiments. The end result is explained by the many combinations of CAPECs that have similar risk scores.

A useful baseline measure we can examine when looking at the individual variants, is the number of unique CAPECs found in a population across the 30 runs. The large and small networks are exposed to the vulnerabilities of 196 and 186 CAPECs (respectively).

`Attack`

The goal of the attack experiments was to examine the performance of an attacker that is competing against a fixed defender. We examine the individual solutions which consist of CAPECs below.

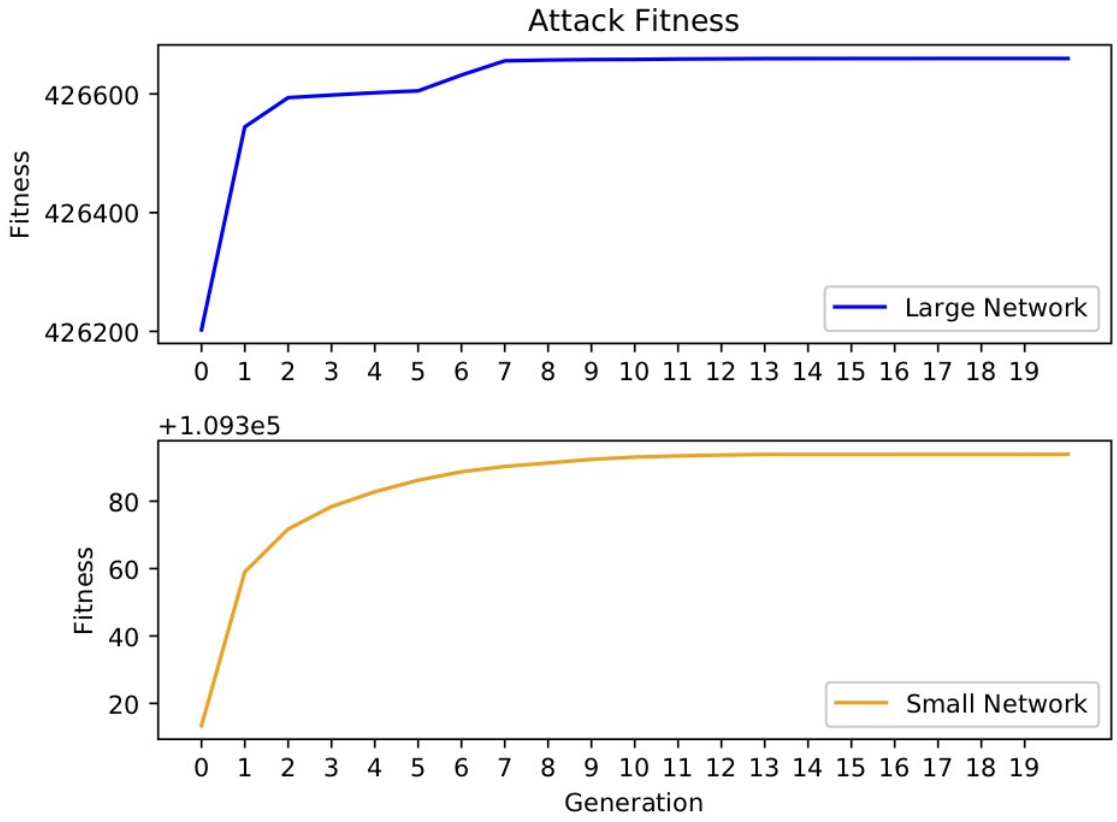

Figure 4-7: Average max fitness values for each generation across 30 runs for the `Attack`evolutionary search.

Table 4.5: Most frequent attack patterns that occurred in the best performing individual in the large network Attack experiment across the 30 runs.

Number of CAPECs per group	CAPECS
1	Using Malicious Files (32)
	Manipulating Web Input to File System Calls (37)
	Leverage Executable Code in Non-Executable Files (39)
	XML Oversized Payloads (42)
	Overflow Buffers (61)
2	Manipulating Web Input to File System Calls and Overflow Buffers (33)
	Leverage Executable Code in Non-Executable Files and XML Oversized Payloads (25)
	Restful Privilege Elevation and XML Oversized Payloads (21)
	AJAX Fingerprinting and Overflow Buffers (20)
	Using Malicious Files and XML Oversized Payloads (18)

Attacker Patterns Almost all ($\approx$ .98) of the CAPECs that posed a risk to the networks were found at least once in the best performing individual phenotype across both networks. There were many more unique CAPECs that appeared in the phenotypes: the attacker found a total of 517 and 515 unique CAPEC values across all the individuals for the large and small networks. Since there are only 575 unique CAPECs, the vast majority of possible values for an individual were used at least once during the search, which indicates the importance of population diversity. CAPECs occurred quite frequently compared to others as can be seen Table 4.5. Many of the CAPECs that frequently occur are related to buffer overflow. Another interesting point to note about the CAPECs is that some CAPECs were more likely to occur with certain CAPECs. If the risk score of CAPECs was additive, then we would expect that the two most frequently occurring individual CAPECs, would be the most frequently occurring pair of CAPECs. However, as we see from Table 4.5, this is not the case, indicating that the attack power of combining two CAPECs is more complicated than a simple additive process.

Fitness Scores From Figure 4-7, we see that attackers for both the large and small networks were able to reach the maximum fitness within a few generations. This is expected since the defender is static.

Table 4.6: Nodes selected for mitigation in the `Defence`experiments. The frequency indicates how many times the highest performing (in a trial) individual phenotype contained this node ID. There were 120 total nodes identified.

Network	Node ID	Frequency	Centrality
Small Network	windows_10-ABUsr11	7	0.007)
	windows_10-ABUsr2	9	0.007
	windows_server_2008-AHProxy	13	0.007
	windows_server_2008-AHDNS	15	0.007
	windows_server_2008-JF	26	0.007
Large Network	windows_server_2008-Dfile4	7	0.0012
	windows_server_2008-AHProxy	9	0.0012
	windows_server_2008-Q	13	0
	windows_10-MCN	22	0.0025
	windows_10-CEF	26	0.012

Table 4.7: Details on the different mitigations that occurred in the best performing individual in the Defense experiments for the large and small networks

Domain	Large Network	Small Network
Client	0.43	0.50
Server	0.57	0.50
OS	0.59	0.64
Sensor_ Add	0.225	0.22
Patch	0.18	0.14
Most Common OS	Windows Server 2008	Windows Server 2008
Most Common Application	Google Chrome	Adobe Acrobat

`Defence`

The goal of the Defense experiments was to examine the performance of a defender against a fixed attacker. In particular, we are interested in how the defender selects nodes to be mitigated since the nodes that are selected would likely be good places for sensor placement. The nodes can be seen in Table 4.6 and the average max fitness values across the 30 trials can be seen in Figure 4-8.

For each mitigation, the defender had several options. The defender could choose to mitigate an 'os' or an 'app' and if they chose to mitigate an app, then the two mitigation options were patch and sensor_add. Additionally, the defender had to chose between updating a client node or a server node. An overview of the different mitigations can be seen in Table 4.7. Generally, we see that the defender chose to

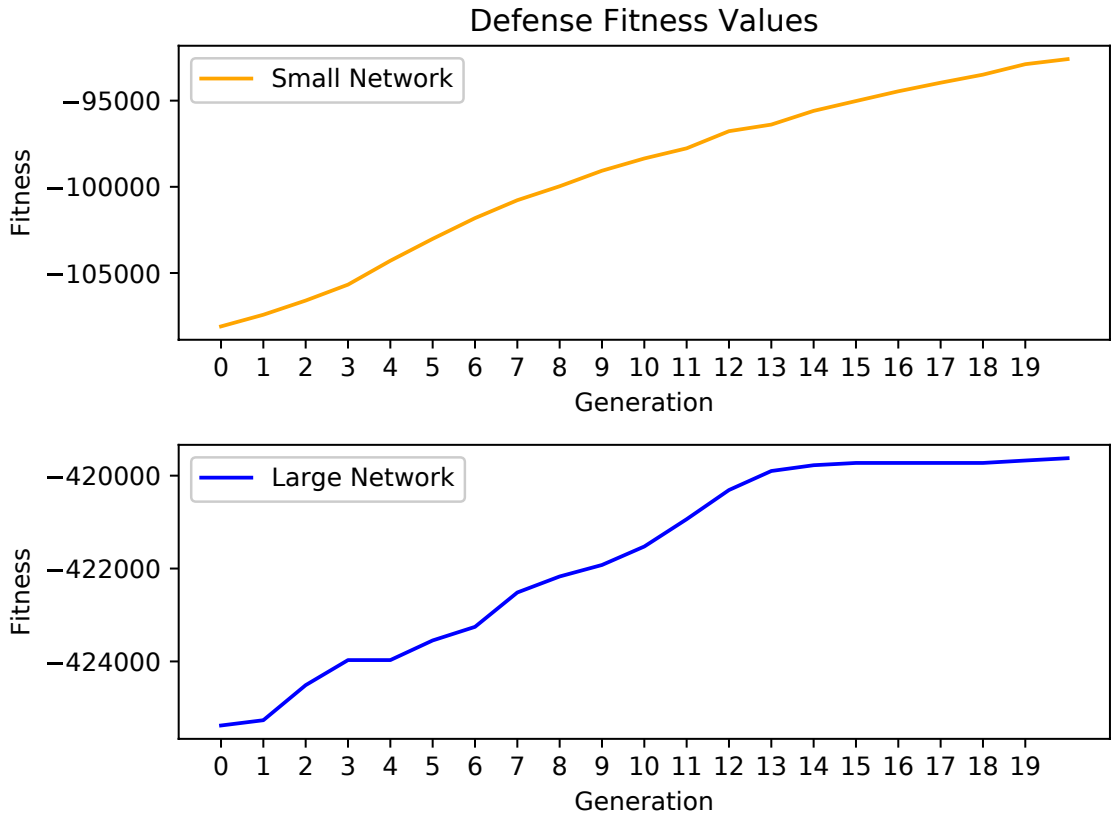

Figure 4-8: Average max fitness values for each generation across 30 runs for the Defense evolutionary search

Table 4.8: Details on the different mitigations that were found in the best performing defender for the large network in the co-evolution experiments. The numerical values indicate the percentage of the mitigations that contained this value.

Domain	Recent_Coev	Coev	LS_Coev
Clients	0.33	0.55	0.56
Servers	0.667	0.45	0.44
OS	0.54	0.53	0.58
Sensor _Add	0.23	0.275	0.25
Patch	0.23	0.195	0.17
Most Common OS Mitigated	Widows 10	Windows 10	Windows 10
Most Common App Mitigated	Internet Explorer	Adobe Acrobat	Adobe Acrobat

mitigate the OS most of the time and that the OS was usually a Windows 2008 Server. We also see that for applications, Sensor _ Add was more frequent than Patch, which makes sense since Sensor _Add has a higher impact. The two networks differed in the application that was most mitigated. However, both also contained many applications that were mitigated only once during the experiments.

Coevolving defender and attacker

A summary of the co-evolution fitness values found during training can be seen in Table 4.3. Summaries of the different mitigations found by co-evolution defenders can be found in Tables 4.8 and Table 4.9. Graphs of the fitness values can be seen in Figure 4-9.

Recent_Coev From the out-of-sample performance, we see that in the defense case Recent_Coev co-evolution, has better performance than the static cases but worse performance compared to Coev and LS_Coev. The algorithm was able to find 468 unique CAPECs across the 30 runs in both the large network and small network. However, the attacker only found 91% of the possible CAPECs for both networks which is smaller compared to the percentage found by the attacker evolving against a fixed defender. This is expected because adapting to an adapting adversary is more challenging for evolutionary search.

The defense mitigations used by Recent_Coev were a bit different than the ones chosen in the `Defence`experiments but were fairly consistent with the mitigations

Table 4.9: Co-Ev nodes selected for mitigation. The frequency indicates how many times the highest performing (in a trial) individual phenotype contained this node ID. There were 120 total nodes identified

	Large Network			Small Network		
Type of Co-Evolution	**Node ID**	**Frequency**	**Centrality**	**Node ID**	**Frequency**	**Centrality**
Recent_Coev	ABFile	8	0.0012	AGF	9	0.007
	CFTP	9	0.0012	windows_server_2008 AHDNS	12	0.007
	DFile 3	12	0.0012	windows_server_2008 AIDNS	21	0.007
	DFile 4	16	0.0012	JF	21	0.007
	windows _10 CEF	35	0.0012	AIF	25	0.007
Coev	CK	3	0.0012	windows_10-ABUsr15	3	0.007
	windows_10 CL	3	0.0012	windows_server_2008 -OFDNS	3	0.007
	windows_10 CK	4	0.0012	windows_server_2008- AIDNS	4	0.007
	windows_10 MCN	7	0.0025	windows_server_ 2008 0OWeb	4	0.007
	windows_10 CEE	7	0.0012	windows_server_ 2008 A	5	0.007
LS_Coev	windows_10 FileCont	3	0.0012	windows_server_2008 AGF	2	0.007
	windows_server_2008 DFile4	3	0.0012	windows_server_2008 AINC File	2	0.007
	windows_10 CK	4	0.0012	windows_10 ARUser14	3	0.007
	windows_10 MCN	6	0.0025	windows_server_2008 AHproxy	4	0.007
	windows_10 CEF	8	0.0012	windows_server_2008 JF	10	0.007

Table 4.10: Attack patterns that occurred in the best performing individual in the small and large network co-evolution experiment across the 30 runs

Type of Co-Evolution	Large Network	Small Network
Recent_Coev	Overflow Buffers (20)	Overflow Buffers (19)
	Dom-Based XSS (14)	Leverage Executable Code in Non-Executable Files (13)
	XML Oversized Payloads (12)	Manipulating Web Input to File System Calls (13)
	XSS Using MIME Type Mismatch (12)	Target Programs with Elevated Privileges (12)
	Buffer Overflow in an API Call (11)	XML Oversized Payloads (12)
Coev	Using Malicious Files (11)	Target Programs with Elevated Privileges (9)
	Filter Failure through buffer overflow (11)	Restful Privilege Escalation (11)
	Restful Privilege Escalation (12)	Buffer Overflow (12)
	Overflow Buffers (12)	Using Malicious Files (12)
	XML Oversized Payloads (15)	XML Oversized Payloads (16)
LS_Coev	XML Oversized Payloads (11)	Leverage Executable Code in Non-Executable Files (10)
	Manipulating Web Input to File System Calls (12)	AJAX Fingerprinting (10)
	XML Nested Payloads (14)	XML Oversized Payloads (11)
	Overflow Buffers (15)	XML Nested Payloads (15)
	Dom-Based XSS (17)	Overflow Buffers (16)

found by Coev and LS_Coev. Recent_Coev preferred servers to clients which was different than Coev and LS_Coev. Further analysis is required to figure out why.

Coev Coev had the second best performance out the algorithm variants in the out-of-sample test. We see that similar attack patterns were chosen for the solutions. There were 475 and 482 unique CAPEC individuals for the large and small networks, which accounted for about 93% of the possible CAPECs from both networks. One of the main benefits of co-evolution is the ability to explore a more diverse group of both defenders and attackers which ultimately leads to being more 'prepared' for an unknown attacker. Finding 93% of the possible attackers combined with the varied defenders led to the increased performance over the Recent_Coev.

LS_Coev LS_Coev had the highest fitness in the out-of-sample defender test. While the most frequently used CAPECs were similar, the number of unique CAPECs found and the percentage of possible CAPECs was higher for LS_Coev: 482 unique CAPECs for the large network, which account for 94% of the possible CAPECs of the large network. While the increase is not large, there are some individual CAPECs that can be quite damaging and can lead to a large increase or decrease in fitness. If we increased the number of lock step generations, we likely would get close to having at least one of each possible CAPEC in an attack solution. LS_Coev and Coev

used mitigations similarly.

4.2.3 General Discussion

From the plots in Figure 4-9, we can see the two sides of the co-evolution responding to the increasing and decreasing strengths. In particular, we see that early in Lockstep co-evolution, the attack gets particularly strong (from the extra attack evolutions). This causes a corresponding decrease in fitness for the lockstep defender. While the training defender fitness for LS_Coev is lower than for Coev, the increased attacker strength, ultimately helps the LS_Coev perform better in the test fitness evaluation.

From just examining which nodes were selected for mitigation, it would be difficult to determine which nodes should be used for sensor placement. However, from the details of the mitigations, we see that generally, both alternating and lockstep co-evolutions, mitigated the same OS and applications. The fact that were not the exact same nodes were not chosen was likely due to the fact that the nodes on the networks are fairly similar.

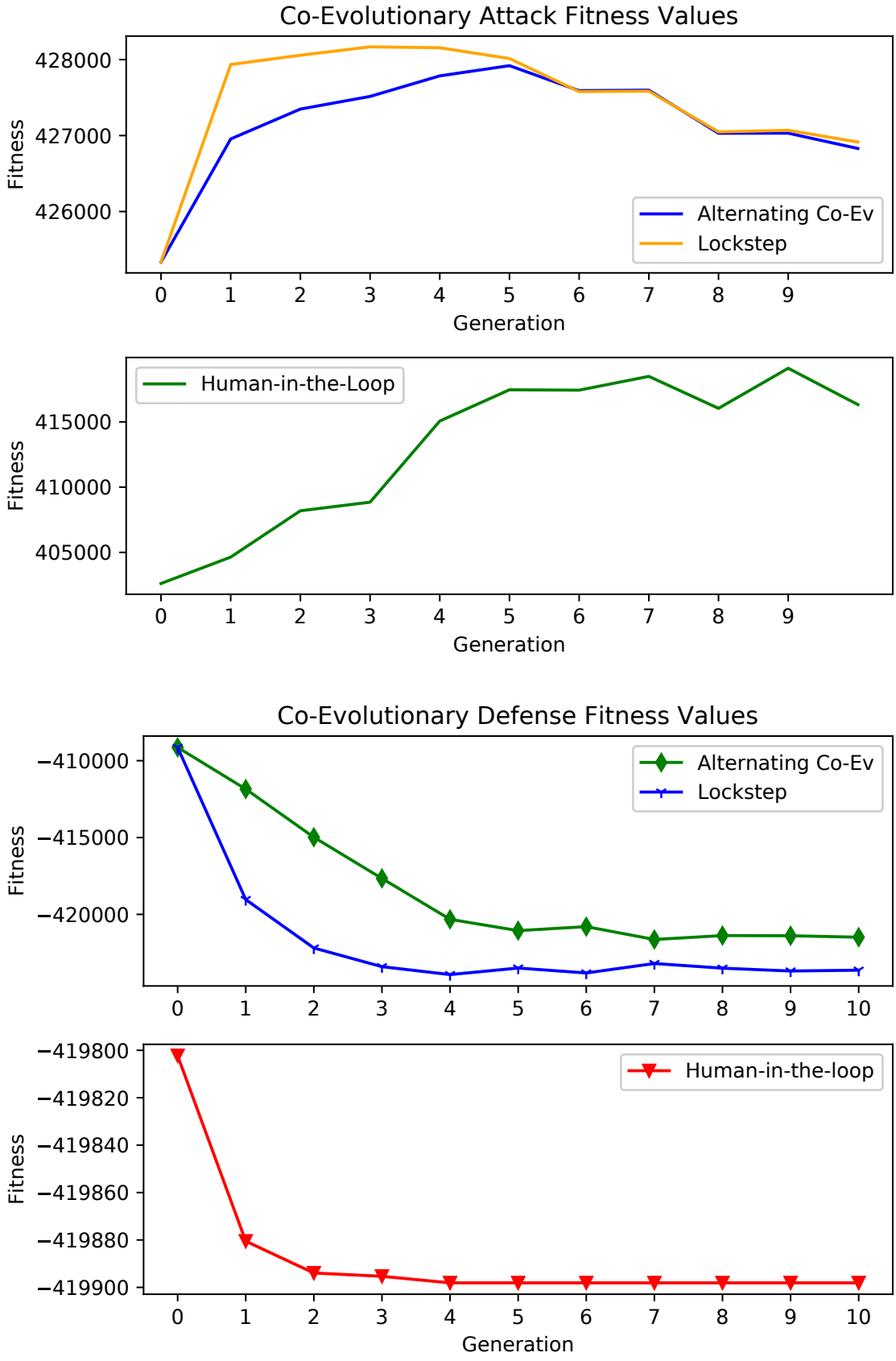

Figure 4-9: Fitness values over time during the co-evolutionary searches.

Chapter 5

Conclusion and Future Work

In this thesis, we presented two systems **BRON** and **CHUCK** that enable cyber analysts and large organizations to participate in a form of active cyber hunting that uses current public threat knowledge and takes into account an unknown attacker. **BRON** demonstrated how a centralized relational schema of public threat data can be used to identify known known attack patterns and vulnerabilities that a network, such as one at a hospital, is vulnerable too. Additionally, we showed how a meta-analysis of existing public threat data points to areas that need in existing public threat data. Information on the quality of public threat data can help organizations with large networks decide what threat to focus on. In **CHUCK**, we described how to build an automated system for sensor placement that combines co-evolutionary algorithms and existing public threat data. **CHUCK** can be quite helpful for large organizations like hospitals that are constantly facing new threats since provides decision support for sensor placement that takes into account an evolving adversary. We also identified co-evolutionary algorithm variant performed best in our experiments. What is novel about our approach in **CHUCK** is that by using evolutionary and co-evolutionary algorithms, we were able to take into account an evolving adversary as well as known vulnerabilities in a network.

Future work would likely take the most vulnerable applications as determined by our system and then place a sensor in that location. Once the sensor is in place, we could run more co-evolutionary experiments to see if an adversary can find ways to

avoid the sensor and to see if the sensor placement should be changing. Another area of future work would be to test the system on more realistic topologies, such as ones in hospitals. This could give a better idea how the system would perform on actual network.

Bibliography

[1] Andy Applebaum, Doug Miller, Blake Strom, Chris Korban, and Ross Wolf. Intelligent, automated red team emulation. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 363–373, 2016.

[2] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[3] R Danyliw, J Meijer, and Y Demchenko. The Incident Object Description Exchange Format. Technical report, RFC Editor, 2007.

[4] Jessica Davis. Ransomware Attacks Double in 2019, Brute-Force Attempts Increase, September 2019.

[5] Distributed Artificial Intelligence Laboratory. NeSSi.

[6] Josiah Dykstra and Celeste Lyn Paul. Cyber operations stress survey (coss): Studying fatigue, frustration, and cognitive workload in cybersecurity operations. In *11th {USENIX} Workshop on Cyber Security Experimentation and Test ({CSET} 18)*, 2018.

[7] Fire Eye. Indicators of Compromise (IOC).

[8] Dennis Garcia, Anthony Erb Lugo, Erik Hemberg, and Una-May O'Reilly. Investigating coevolutionary archive based genetic algorithms on cyber defense networks. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pages 1455–1462, New York, NY, USA, 2017. ACM.

[9] Steven Gianvecchio, Christopher Burkhalter, Hongying Lan, Andrew Sillers, and Ken Smith. Closing the gap with apts through semantic clusters and automated cybergames. In *International Conference on Security and Privacy in Communication Systems*, pages 235–254. Springer, 2019.

[10] Guardicore. Infection Monkey.

[11] Erik Hemberg, Joseph R. Zipkin, Richard W. Skowyra, Neal Wagner, and Una-May O'Reilly. Adversarial co-evolution of attack and defense in a segmented computer network environment. In *Proceedings of the Genetic and Evolutionary*

Computation Conference Companion, GECCO '18, pages 1648–1655, New York, NY, USA, 2018. ACM.

[12] Sushil Jajodia and Steven Noel. Topological vulnerability analysis. In *Cyber situational awareness*, pages 139–154. Springer, 2010.

[13] Jonathan Kelly, Michal Shlapentokh-Rothman, Erik Hemberg, Nick Rutar, and Una-May O'Reilly. Bron - bridging public threat data for cyber hunting. 2019.

[14] Sitaram Kowtha, Laura A Nolan, and Rosemary A Daley. Cyber security operations center characterization model and analysis. In *2012 IEEE Conference on Technologies for Homeland Security (HST)*, pages 470–475. IEEE, 2012.

[15] Information Technology Laboratory. Vulnerability metrics cvss.

[16] Luatix. Open Cyber Threat Intelligence Platform.

[17] Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1795–1812, 2019.

[18] MITRE. Att&ck matrix for enterprise.

[19] MITRE. BRAWL Automated Adversary Emulation Exercise.

[20] MITRE. Caldera: Automated Adversary Emulation.

[21] MITRE. Common attack pattern enumeration and classification.

[22] MITRE. Common platform enumeration.

[23] MITRE. Common vulnerabilities and exposure.

[24] MITRE. Common weakness enumeration.

[25] Pat Muoio and Paul Green. Open Command and Control (OpenC2). Technical report, G2, 2015.

[26] Steven Noel and Sushil Jajodia. Optimal ids sensor placement and alert prioritization using attack graphs. *Journal of Network and Systems Management*, 16(3):259–275, 2008.

[27] Joseph Pamula. Attack graphs: scalable construction and analysis. 2007.

[28] Rich Piazza, John Wunder, and Bret Jordan. STIX™ Version 2.0. Part 1: STIX Core Concepts, July 2017.

[29] Elena Popovici, Anthony Bucci, R Paul Wiegand, and Edwin D De Jong. Coevolutionary principles., 2012.

[30] A. Razaque, F. Amsaad, M. Jaro Khan, S. Hariri, S. Chen, C. Siting, and X. Ji. Survey: Cybersecurity vulnerabilities, attacks and solutions in the medical domain. *IEEE Access*, 7:168774–168797, 2019.

[31] Red Canary. Atomic Red Team.

[32] Mohsen Rouached and Hassen Sallay. An Efficient Formal Framework for Intrusion Detection Systems. *Procedia Computer Science*, 10:968–975, December 2012.

[33] George Rush, Daniel R. Tauritz, and Alexander D. Kent. Coevolutionary agent-based network defense lightweight event system (candles). In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 859–866, New York, NY, USA, 2015. ACM.

[34] Reginald E Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *European Symposium on Research in Computer Security*, pages 18–34. Springer, 2008.

[35] Blake E Strom, Joseph A Battaglia, Michael S Kemmerer, William Kupersanin, Douglas P Miller, Craig Wampler, Sean M Whitley, and Ross D Wolf. Finding cyber threats with att&ck-based analytics. Technical report, MITRE, 2017.

[36] Zareen Syed, Ankur Padia, M. Lisa Mathews, Tim Finin, and Anupam Joshi. UCO: A Unified Cybersecurity Ontology. AAAI Press, February 2016.

[37] T. Tervoort, M. T. De Oliveira, W. Pieters, P. Van Gelder, S. D. Olabarriaga, and H. Marquering. Solutions for mitigating cybersecurity risks caused by legacy software in medical devices: a scoping review. *IEEE Access*, pages 1–1, 2020.

[38] Ann Thorhauer and Franz Rothlauf. On the locality of standard search operators in grammatical evolution. In *International Conference on Parallel Problem Solving from Nature*, pages 465–475. Springer, 2014.

[39] Uber. Metta.

[40] Verizon Security Research & Cyber Intelligence Center. Vocabulary for Event Recording and Incident Sharing, 2013.

[41] Cynthia Wagner, Alexandre Dulaunoy, Gérard Alexandre, and Andras Iklody. MISP: The Design and Implementation of a Collaborative Threat Intelligence Sharing Platform. pages 49–56, October 2016.

[42] Ju An Wang and Minzhe Guo. OVM: An Ontology for Vulnerability Management. April 2009.

[43] Richard Winton. Hollywood hospital pays $17,000 in bitcoin to hackers; FBI investigating. *The Los Angles Times*, February 2016.