University of Nevada, Reno

Cybersecurity Information Exchange with Privacy (CYBEX-P) and TAHOE – A Cyberthreat Language

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computer Science and Engineering

by
Farhan Sadique

Dr. Shamik Sengupta/Dissertation Advisor

DECEMBER 2021

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

FARHAN SADIQUE

entitled

Cybersecurity Information Exchange with Privacy (CYBEX-P) and TAHOE - A Cyberthreat Language

be accepted in partial fulfillment of the
requirements for the degree of

Doctor of Philosopy

Shamik Sengupta, Ph.D.,
Advisor

Engin Arslan, Ph.D.,
Committee Member

Shahriar Badsha, Ph.D.,
Committee Member

Haoting Shen, Ph.D.,
Committee Member

Hanif Livani, Ph.D.,
Graduate School Representative

David W. Zeh, Ph.D., Dean
Graduate School

December, 2021

Abstract

Cybersecurity information sharing (CIS) is envisioned to protect organizations more effectively from advanced cyberattacks. However, a completely automated CIS platform is not widely adopted. The major challenges are: (1) the absence of advanced data analytics capabilities and (2) the absence of a robust cyberthreat language (CTL). This work introduces Cybersecurity Information Exchange with Privacy (CYBEX-P), as a CIS framework, to tackle these challenges. CYBEX-P allows organizations to share heterogeneous data from various sources. It correlates the data to automatically generate intuitive reports and defensive rules. To achieve such versatility, we have developed TAHOE - a graph-based CTL. TAHOE is a structure for storing, sharing, and analyzing threat data. It also intrinsically correlates the data. We have further developed a universal Threat Data Query Language (TDQL). In this work, we propose the system architecture for CYBEX-P. We then discuss its scalability along with a protocol to correlate attributes of threat data. We further introduce TAHOE & TDQL as better alternatives to existing CTLs and formulate ThreatRank - an algorithm to detect new malicious events.

We have developed CYBEX-P as a complete CIS platform for not only data sharing but also for advanced threat data analysis. To that end, we have developed two frameworks that use CYBEX-P infrastructure as a service (IaaS). The first work is a phishing URL detector that uses machine learning to detect new phishing URLs. This real-time system adapts to the ever-changing landscape of phishing URLs and

maintains an accuracy of 86%. The second work models attacker behavior in a botnet. It combines heterogeneous threat data and analyses them together to predict the behavior of an attacker in a host infected by a bot malware. We have achieved a prediction accuracy of $85 - 97\%$ using our methodology. These two frameworks establish the feasibility of CYBEX-P for advanced threat data analysis for future researchers.

Acknowledgements

I would like to thank the following:

- Dr. Shamik Sengupta for his support and direction;
- All the members of the CYBEX-P project for their work, insightful comments and suggestions.
- My parents and families for their unwavering support and belief in me;
- And my friends for a cherished time spent together in social settings.

Contents

List of Figures

Chapter 1

Introduction

The word-wide-web plays a major role in modern life. It offers numerous functionalities including social networking, banking, and e-commerce. The primary reason for the widespread adoption of the internet is the flexibility it offers. For instance, we are presently able to perform major financial operations from the convenience of our homes. However, such flexibility comes at a price in the form of cybercrimes.

Modern cyberspace is beleaguered with an increasing number of advanced cyber-attacks rendering conventional cybersecurity measures practically useless. This is evident from the tenfold increase in the number of data breaches over the past 12 years- from about 160 in 2005 to about 1600 in 2017 [1]. This is also seen in the tenfold increase in number of breached records from about 16 millions in 2010 to about 156 millions in 2020. Furthermore, the Center for Strategic and International Studies estimates the current global cost of cybercrime to be $1 trillion in 2020 seeing

an increase of more than 50% in two years from $600 billion in 2018 [2]. These data show that traditional cybersecurity measures are failing to curb the cybercrimes and demonstrate the need for the adoption of newer agile cybersecurity measures by organizations.

1.1 Motivation

We begin this section with some notable cyberattacks from recent years. In October 2021 the entire code base of Twitch, a video streaming website, was breached and posted online via a torrent link [3]. In 2014, a cyberattack on JPMorgan Chase compromised the accounts of 76 million households and 7 million small businesses [4]. The attack began in June and remained undiscovered until July. By the time the bank's security team discovered the breach, the hackers had reached more than 90 servers with complete access to dozens of servers. The same hackers also targeted 13 other financial firms Citigroup, HSBC Holdings, etc. [5]. They were able to breach another financial institute called Fidelity Investments as part of the same campaign [6].

In 2013, hackers breached the Target Corporation's network to steal 40 million credit and debit card numbers [7]. The credit unions suffered a total cost of over 200 million dollars just to reissue those cards. The data breach was initiated by compromising a vendor's system through a simple phishing attack [8]. In 2013, Yahoo suffered

from a data breach compromising data of 1bn users [9]. In 2017, Yahoo changed the estimate to 3bn, which means every single Yahoo account was breached [10]. In 2014, further 500 million accounts were stolen from Yahoo [11]. In 2017, a ransomware named Wannacry crippled the United Kingdom's National Health Service (NHS) by exploiting a vulnerability in Microsoft Windows XP [12].

There are several noticeable takeaways from the above case studies. Firstly, all kinds of industries including healthcare, finance, retail, etc. are susceptible to cyberattacks. Secondly, Even the largest organization is unable to defend itself against all kinds of cyberattacks. This claim is supported by the fact that JPMorgan spent $250 million on cybersecurity annually, yet they could not prevent the data breach. Thirdly, the attacks are not always isolated. As discussed earlier, the same hacker group attacked 14 financial institutes during the 2014 JP Morgan Chase breach and was able to hack 2 of them. It is apparent that all these institutes could benefit in the future with some form of collaboration in cybersecurity. Finally, even the major attacks can be easily prevented, as the Target breach started from a simple phishing attack.

It is evident from the above case studies that conventional cybersecurity measures are unable to fend off modern cyberattacks. The risk is exacerbated by the emergence of organized and state-backed cyberattack campaigns, which have left even big firms practically defenseless [13]. Consequently, a new approach towards cybersecurity is essential to thwart organized cyber-attack campaigns proactively, to anticipate and mitigate large-scale exploitation, and to respond faster to emerging cyber threats.

Cybersecuritiy Information Sharing Collaborative cybersecurity information sharing (CIS) is envisioned to protect organizations more effectively from advanced cyberattacks [14]. The benefits of proactive sharing are twofold — (1) new threats are detected faster, owing to collaborative analysis (2) the corresponding signatures are distributed faster due to real-time sharing. Widespread participation in CIS will potentially accelerate the signature development process. Furthermore, real-time sharing of this information via a centralized platform will play a key role in disseminating those signatures to interested parties. Besides, a CIS will have access to a great variety of threat data from diverse sectors. The diversity of threat data along with the sheer volume is will make a CIS better at correlation, pattern analysis, and threat detection. Furthermore, the US Congress proposed a number of acts incentivizing private organizations and requiring public organizations to share threat data [15, 16].

1.2 Challenges

Despite all the benefits, there is limited sharing in the industry. Several proprietary CIS frameworks have emerged in the industry including ThreatConnect [17], AlienVault [18], X-Force [19], ThreatStream [20], ThreatExchange [21], and EclecticIQ [22]. However, all of them have failed to attract wide-scale participation because of several limitations of existing platforms:

1. The existing platforms are built for only data sharing [23] or for limited data analysis [24], although robust data analysis is just as important [25],

2. They cannot generate actionable CTI from machine data [24, 26]; automatic data collection is either absent [27] or limited [28, 29] in existing platforms, although complete automation is expected [26, 30, 31],

3. There is no standard cyberthreat language for all of data sharing, storing, correlation, and analysis [26, 32, 33]

4. There are no intuitive investigation tools [25], and

5. The existing platforms cannot outline a defensive course of action - e.g. automatic generation of firewall rules, although it is desired in a complete CIS ecosystem [34].

There is not one existing CIS platform, which tackles all of the above challenges.

1.3 Functions of a CIS platform and threat models

In light of the above challenges, we have outlined several functions of a central information sharing platform (CIS). These functionalities and associated threat models are described below:

Data forwarding Many CIS platforms [23, 24] adopt a publisher-subscriber architecture. In this design, the CIS platform acts as an information broker only. The threat model in this scenario considers the CIS platform to be semi-trusted (known also as honest-but-curious). All the subscribers and publishers are assumed to be trusted. Under this simplified premise, the privacy problem essentially becomes hiding the data from the CIS platform during sharing.

Data analysis The CIS platform can correlate and analyze. The analysis can be anything from simple aggregation to learning a complex model. This paradigm considers the privacy-preserving analysis of the shared data. The CIS platform should be able to analyze the stored data and present the analysis reports without revealing any sensitive information about the sharers.

The threat model considers all the parties including the sharing platform to be untrusted. Therefore, the CIS platform is only allowed to share aggregations or learned models which do not contain any sensitive raw data. This is a much more difficult problem to solve. The effective solution to this problem is expected to result in widespread participation in collaborative cyber intelligence.

Dispute resolution A participant may share wrong or malicious data with the CIS platform. Other participants should be able to dispute such data without knowing the identity of the origin. The CIS platform should be able to resolve such disputes without revealing any sensitive information regarding the origin of the data.

Statistical Disclosure Controls (SDC) This threat model considers a group of direct identifiers (name, phone number, etc.) that directly identify a person and a group of indirect identifiers (time of activity, size of the network, etc.) which when analyzed together may reveal the identity of the owner. Let us consider a set of n secondary identifiers $S = s_1, ..., s_n$. In this threat model, we are concerned with $S' \subseteq S$ that may reveal a direct identity of a person or an organization.

SDC is often used by surveyors in data-driven research to hide the identities from analysis results. SDC ensures that no person or organization can be identified from the release of microdata. The same techniques should be applied to a CIS platform to hide the identities of the participants from the aggregate data.

1.4 Thesis Contribution

CYBEX-P: To tackle these challenges, we introduce CYBersecurity information EXchange with Privacy (CYBEX-P) in this research work. CYBEX-P is a cybersecurity information sharing (CIS) platform with robust data governance. It automatically collects and correlates heterogeneous threat (e.g. malware, firewall log, etc.) data from different organizations to generate insightful reports.

CYBEX-P sets itself apart from the existing frameworks by providing a completely automated robust framework for data collection, data analysis, and report generation. A major contribution of CYBEX-P lies in merging multiple types of threat

data from various organizations and analyzing them together. These features make CYBEX-P a versatile and robust information sharing framework suitable for all types of organizations, small or big, public or private.

TAHOE: Any CIS platform like CYBEX-P potentially handles hundreds of different data formats. Thus, it needs a standard data format and structure to represent threat data. A cyberthreat language (CTL) is a specification of how to format and serialize any kind of threat data. To that end, we introduce TAHOE - a graph-based cyberthreat language (CTL) for storing, sharing, analyzing, and intrinsically correlating data.

TAHOE automatically correlates attributes of incoming threat events with older ones. In contrast to traditional CTLs like STIX[35] or MISP format [36], TAHOE is built from ground-up for faster data correlation and blind processing.It also lays the foundation for ThreatRank - a novel graph-based algorithm to detect previously unseen malicious events using correlation. CYBEX-P uses TAHOE for data storage, sharing, and analysis.

The rest of this dissertation is organized as follows: Chapter 2 discusses previous works related to this research. It highlights how CYBEX-P and TAHOE are different from existing CIS frameworks and CTLs. Chapter 3 introduces the system architecture of CYBEX-P and familiarizes the reader with the flow of data through different parts of CYBEX-P. Chapter 4 introduces TAHOE as a novel structure to represent

threat data. We compare TAHOE with traditional CTLs and show how it facilitates data analysis and correlation. Chapter 5 presents a novel real-time phishing URL detection framework that uses CYBEX-P infrastructure as a service (IaaS). Chapter 6 presents another novel framework that sits on top of CYBEX-P and models attacker behavior in a botnet. Finally, Chapter 7 concludes the dissertation and puts forward a path for future work.

Chapters 5 and 6 both introduce two novel threat analysis mechanisms. In this dissertation, they serve the key purpose of showing how CYBEX-P provides infrastructure as a service (IaaS) for advanced threat data analysis. The work in chapter 5 works with a specific type of threat data namely URLs. On the other hand, the work in chapter 6 takes a more robust approach. It merges several types of threat data and analyzes them together to model attacker behavior. Our experimental results in this works show that CYBEX-P is scalable and suitable for real-time threat analysis. Thus in this paper, we conclude that CYBEX-P can disrupt the rapid and extensive spread of new threats.

Chapter 2

Related Work

Having introduced the motivations and challenges, in this chapter we discuss the previous works related to cybersecurity information sharing. We discuss works from both academia and the industry. We focus on information sharing, and network forensics in separate sections. Finally, we compare our research with the most popular alternatives in terms of contribution and novelty.

2.1 Cybersecurity Information Sharing

While there are plenty of works on cybersecurity information sharing (CIS), none of them provide a comprehensive solution to the aforementioned challenges. In this section we discuss these works, focusing on the CIS frameworks.

We begin our study with CIS frameworks proposed in academia. Edwards et al. [37] presented one of the earliest frameworks for sharing vulnerability information. Another framework was presented by Zhao et al. [38] for collaborative information sharing. Yet another framework, called SKALD, [39] was developed by Webster et al., for real-time sharing. However, none of these provide capabilites for advanced analysis of heterogeneous threat data. CYBEX-P sets itself apart from these early works by providing a robust system architecture along with different modules of for correlating and analyzing hetegrogeneous threat data.

Gordon et al. [40] show that information sharing lowers a firm's overall cyber investment. However, they identified the limitation of the mechanism includes the absence of economic incentives to promote effective information sharing. Particularly it does not address the free-riding behavior and possible malicious intent of the participants.

Gordon et al. [41] focus on an associated problem - the tendency of firms to not invest in cybersecurity until a breach occurs. They show that information sharing reduces the uncertainty associated with the decision to invest in cybersecurity. As a result, it is in the best interest of the firms to make cybersecurity investments sooner than later.

Hernandez et al. [42] formulate information sharing as a risk-based decision-making model. The authors represent the sharing community as directed graphs, with nodes representing participants and edges representing relationships. They argue that such

modeling will aid in dealing with several problems associated with such a platform-especially the cost-incentive analysis.

The [43] presents a novel mechanism to represent raw cyber threat data in Structured Threat Information Expression (STIX) [35] format in an automated manner. This work classifies the data into 4 levels based on sensitivity: Level 0 data is not sensitive at all, level 1 data is highly sensitive and masked using hashing, level 2 data is masked using encryption and can be shared on-demand, and level 3 data is shared to only find out if anyone else received similar data from other sources. The [44] builds on the previous work proposes a complete cybersecurity information sharing framework called CYBersecurity information EXchange with Privacy (CYBEX-P).

Various protocols, specifications, and implementations such as TAXII, OpenIOC, VERIS, MAEC, SCAP, and IODEF have also been developed to provide a common platform for sharing cybersecurity information [24, 45–47].

A large number of these works [48–51] took a game-theoretic approach by modeling the cybersecurity information sharing as a 3-player game. Figure 2.1 depicts cyber-security information sharing as a 3-player game.

While the 3-player modeling is more popular in literature, some authors [52] have considered a fourth player: data analyst. Data analyst or data researcher in a broad sense analyzes the data to come up with meaningful insights. The figure 2.2 shows such a model.

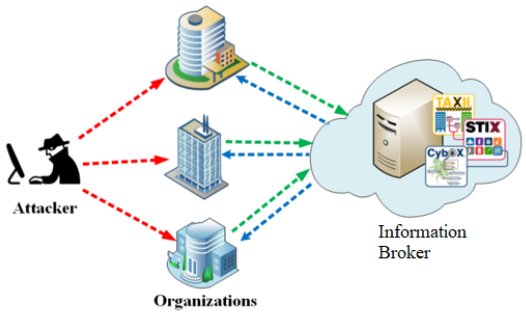


FIGURE 2.1: Cybersecurity information sharing as a 3-player game

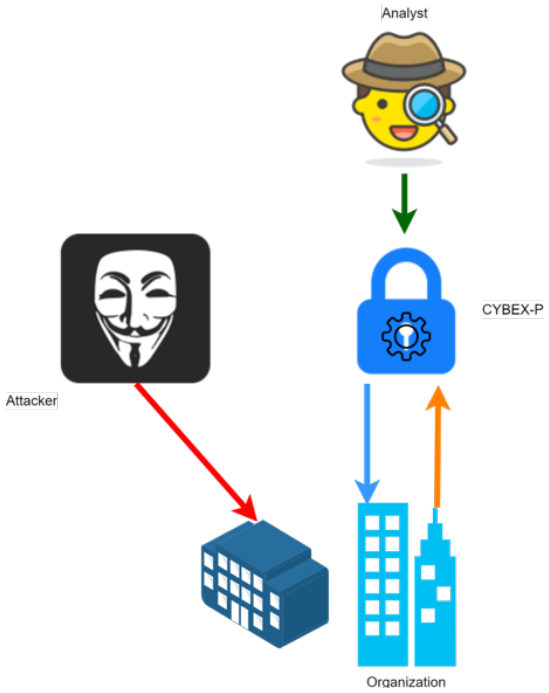


FIGURE 2.2: Cybersecurity information sharing as a 4-player game

Tosh et al. [48] propose an incentive model that enforces the firms to share information truthfully. They perform theoretical analysis to find the conditions when a firm's utility is maximized concerning its investment. The numerical results in their work show that firms are benefited from sharing cybersecurity information in general.

Tosh et al. [49] introduce an evolutionary game-theoretic framework for cybersecurity information sharing called CYBersecuriy information EXchange (CYBEX). They

show how CYBEX can vary its cost of participation to motivate sharing and at the same time increase its revenue.

Kamhuoa et al. [50] also take a game-theoretic approach to model cybersecurity information sharing between two parties. The key finding of their work is showing that a Nash equilibrium exists and deriving it. The model is however based on several simple assumptions including, two participating users, homogeneous vulnerabilities, and complete information.

Vakilinia et al. [51] take a coalitional game-theoretic approach to calculate rewards and participation fees. They formulate a coalitional game between organizations and analyze the well-known Shapley value and Nucleolus solution concepts in a cybersecurity information sharing system.

Meanwhile, several proprietary frameworks have emerged in the industry including ThreatConnect [17], AlienVault [18], X-Force [19], ThreatStream [20], ThreatExchange [21], and EclecticIQ [22]. All of these suffer from one or more of three major limitations: (1) the data are inputted by humans not automated (2) they do not support advanced threat data analysis (3) they have limited scope in participant or type of data. For example, ThreatExchange does not allow educational institutions, X-Force data are written by humans, and so on. CYBEX-P, on the other hand, is built from the ground up keeping these challenges in mind.

2.2 Network Forensics

Plenty of research has been conducted on network forensics in the past. Corey et al. [53] established the 3 requirements of a network forensics analysis tool (NFAT). Yasinsac et al. [54] outlined the policies which would enable an organization to perform effective network forensics.

Pilli et al. [55] put forward a generic framework for network forensics. Shanmugasundaram et al. [56] introduced ForNet as a distributed NFAT. They outlined the challenges faced during designing an NFAT and the general structure of an NFAT in their work. Cohen [57] introduces yet another forensics tool called PyFlag. PyFlag merges disc, memory network forensics in an open-source tool. However, most of the analysis in PyFlag lives in the upper layers of the OSI model [58] only.

Wang and Daniels [59] were the first to consider a graph-based model for network forensics. A survey of network forensics models and corresponding challenges was conducted by Pilli et al. [60]. Another survey into tools and techniques for network forensics was done by Meghanathan et al. [61]. Khan et al. [62] summarized the taxonomy and open challenges in network forensics in their review paper. Almulhem [63] further clarified the notions and challenges relating to network forensics in his work.

Mukkamala and Sung [64] introduced the use of machine learning in the realm of network forensics to identify significant features. Liao et al. [65] proposed using

fuzzy logic and expert systems for network forensics.

While numerous works have considered different aspects of network forensics, a complete tool for network forensics is yet to be seen. Our present work intends to bridge that gap between research and industry by presenting a user-friendly NFAT.

2.3 Contributions of this Research Work Compared to Previous Works

Now, we move on to the most prominent CIS systems in use today. The primary requisite of any CIS platform is a standardized format or a cyberthreat language (CTL). Presently, the most popular CTL is Structured Threat Information Expression (STIX) [35] developed by the MITRE Corporation. While STIX is suitable for mutual data sharing, in this dissertation we discuss how it is unscalable for any kind of data analysis. We further explain how STIX-based systems are prone to store duplicate data. To overcome these shortcomings, chapter 4 introduces TAHOE - a graph-based CTL for both data sharing and data analysis.

In parallel to STIX, MITRE also developed the Trusted Automated eXchange of Indicator Information (TAXII) [23] protocol to facilitate peer-to-peer data sharing between trusted parties. However, what TAXII gains in data sharing, it lacks in data analysis. Since, TAXII only supports STIX format, it is unsuitable for even the

simplest data analysis. For example, TAXII does not provide any API endpoint to lookup an IP address in its database.

Meanwhile, Wagner et al. developed MISP [24] as a collaborative CTI sharing platform with group-based access control and their own data format. Although database lookups are fast in a MISP server, the open data structure of MISP core format [36] defeats that purpose, because different users structure the same data in different ways. Moreover, MISP structures data in only two levels (Events and Attributes); so representing complex data in MISP format is non-intuitive. A few other limitations of MISP are: it relies heavily on manual human input rather than automating machine data, and it does not provide a robust data governance framework.

In summary, after extensive study, we were primarily inspired by both STIX and MISP data structures while developing TAHOE. However, we have built TAHOE from the ground up with advanced data analytics and speed in mind. Similarly, we were inspired by both TAXII and MISP while designing CYBEX-P. However, CYBEX-P is a complete ecosystem with actionable data rather than a cumbersome tool. As a result, we present TAHOE and CYBEX-P as a perfect marriage between versatility and performance in this dissertation.

Chapter 3

Proposed System - CYBEX-P

Having discussed the existing platforms, in this chapter, we propose CYbersecurity information Exchange with Privacy (CYBEX-P). CYBEX-P is a robust cybersecurity information sharing platform. It collects heterogeneous threat data from different types of organizations and correlates them to generate insightful reports. This chapter familiarizes the readers with the system architecture of CYBEX-P and sets the basis for the novel contributions in subsequent chapters.

3.1 Overview of CYBEX-P

We begin our discussion with a functional overview of CYBEX-P. Fig. 3.1 shows the four major functions of CYBEX-P.

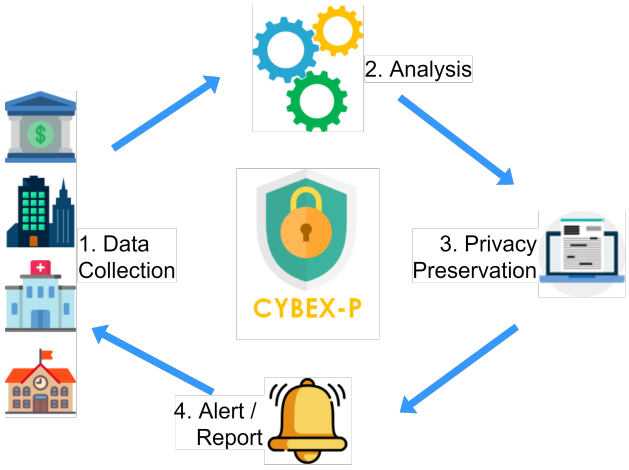


FIGURE 3.1: Overview of the 4 major functions of CYBEX-P.

3.1.1 Data Collection

CYBEX-P is essentially a cloud-based platform for organizations to share heterogeneous cyberthreat data. CYBEX-P accepts all kinds of human or machine-generated data including firewall logs, emails, malware signatures, and handwritten cyberthreat intelligence (CTI).

3.1.2 Data Analysis

In addition to data sharing, CYBEX-P allows the users to correlate and analyze the data. This key feature sets CYBEX-P apart from other cybersecurity information sharing systems.

3.1.3 Privacy Preservation

The second key feature of CYBEX-P is that, the owner/sharer controls, who see which part of the data. We achieve such a granular control by separately encrypting each attribute of the data. This privacy-preservation feature is briefly mentioned in this thesis to make the discussion comprehensive and for the better understanding of the CYBEX-P framework. However, this thesis does not detail the mechanism.

3.1.4 Report/Alert Generation

Finally, users can generate insightful reports or alerts from the data. CYBEX-P also provides a feed of automatically generated defensive (e.g. firewall) rules, as we will discuss in 3.2.5.4. This particular feature reflects our philosophy of making the entire process completely automated.

3.2 System Architecture of CYBEX-P

Now that we have introduced a brief overview of CYBEX-P, in this section, we move deeper into its system architecture. To accommodate the four major functions, we have built CYBEX-P with 6 independent software modules – (1) Frontend, (2) Input, (3) API, (4) Archive, (5) Analytics, and (6) Report. These modules share various components as shown in Fig. 3.2. In addition, we have built a library to manipulate

TAHOE content. TAHOE is a CTL, that CYBEX-P uses, to store, analyze and share data. In this section we explain how this modular architecture provides better manageability of the system as well as better security of the data.

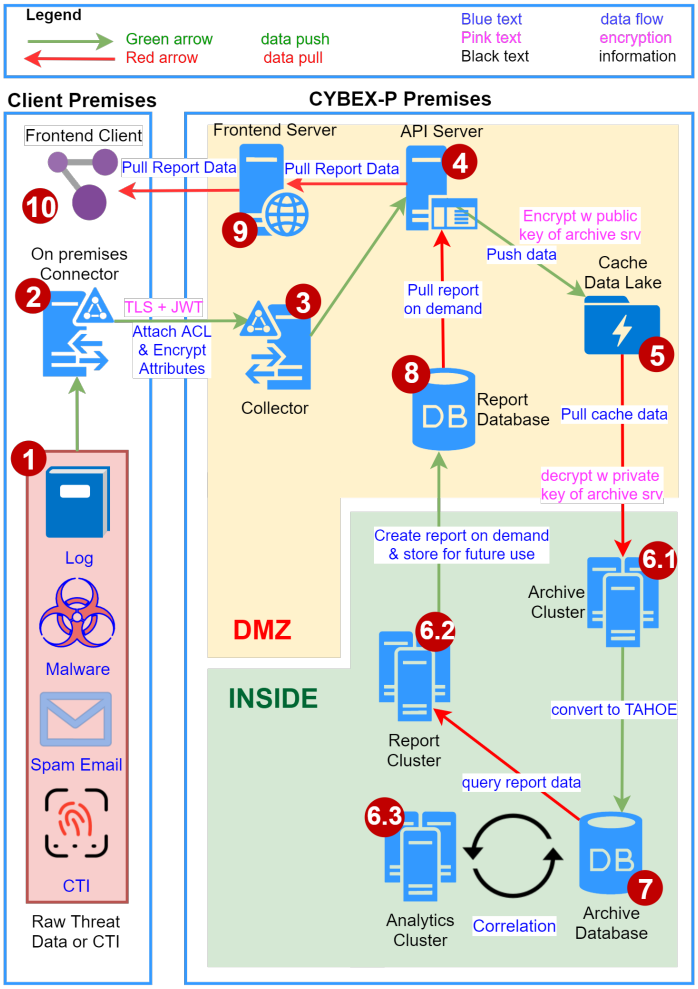


FIGURE 3.2: System architecture of CYBEX-P along with the Data Flow.

3.2.1 Frontend Module

The frontend module (9, 10 in Fig. 3.2) is a webapp for users to interact with CYBEX-P. This module allows users

1. to register with and login to CYBEX-P,
2. to manually upload threat data as text files,
3. to configure machines (e.g. firewalls) to automatically share data with CYBEX-P (explained in 3.2.2),
4. to control the access of their data,
5. to generate and view reports (explained in 3.2.6), and
6. to investigate an incident using our incident investigation tool (explained below).

3.2.1.1 CYBEX-P Threat Intelligence – An Incident Investigation Tool

Fig. 3.3 shows a novel contribution of this project – CYBEX-P Threat Intelligence. This tool, powered by CYBEX-P analytics, allows a user to investigate an incident. This tool uses the CYBEX-P database to get the related attributes. Related attributes are discussed in section 4.1.

Thirdly, CYBEX-P sends a malicious score for all the attributes on the graph. Section 4.2 explains ThreatRank – the algorithm used to calculate these malicious scores.

Finally, the investigation tool colors each attribute blue (unknown), green (benign), yellow (suspicious), or red (malicious) based on the score. A big cluster of red attributes denotes that the original attribute is malicious.

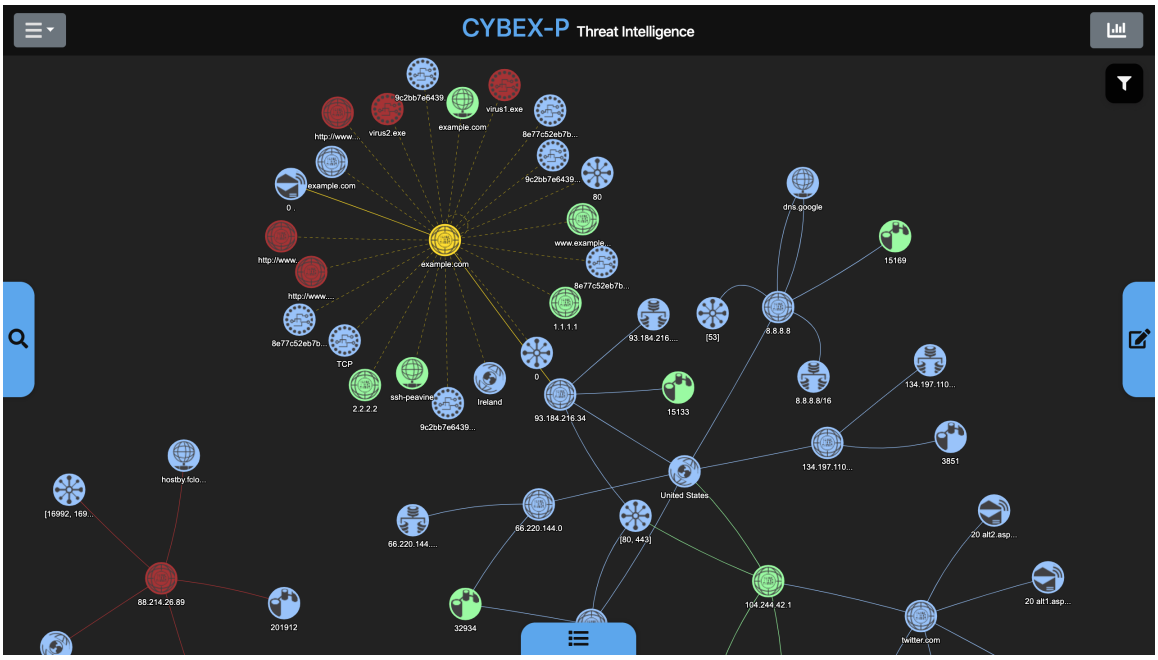


FIGURE 3.3: Graphical Incident Investigation Tool.

3.2.2 Input module

The scope of CYBEX-P spans all kinds of machine-generated threat data. It needs to automatically collect data from various types of devices like firewalls, honeypots, web and email servers, etc. These devices share data in different formats and communicate using a diverse variety of protocols. Example protocols for automatic data collection are: (i) by calling an API, (ii) via a pre-configured websocket, (iii) by reading from a text file, (iv) by reading from a database, (v) using Linux Syslog protocol, etc.

To interface with all these devices, We have developed a robust input module (❶, ❷, ❸, ❹, ❿ in Fig. 3.2). Users can manually upload threat data via a web client (❿) or automatically send machine data via a connector (❷) to the collector (❸).

Afterward, the collector posts the raw data to our API (4) endpoint. To ensure privacy, it uses the transport layer security (TLS) protocol [66] during collection and posting.

3.2.3 API module

The API module (4, 5 in Fig. 3.2) consists of the API server (4) and the cache data lake (5). It acts as the gateway for all data into and out of CYBEX-P. It has two sub-modules –

It serves two primary purposes:

1. The input module (subsection 3.2.2) puts raw data into CYBEX-P via the API.
2. The report module (subsection 3.2.6) sends reports back to users via the API.

3.2.3.1 Data Input sub-module

The input module posts the raw data to the API (4) endpoint. The API encrypts the data with the public key of the archive server (6.1) and stores the encrypted data in the cache data lake (5).

We have placed the API in the demilitarized zone (DMZ) of our firewall because it faces the internet. However, storing data in the DMZ is somewhat risky. So, we encrypt the cache data lake with the public key of the archive server. The archive

server is in the inside zone. This design protects that data even if the DMZ is compromised.

We have also placed the cache data lake in the DMZ so that the API does not initiate a connection from the DMZ to the inside zone. This design – (1) protects the data even if the DMZ is compromised (2) does not require any connection initiation from the DMZ to the inside zone.

3.2.3.2 Report Publishing sub-module

A user can request different reports via the API. The API gets those reports from the report DB (8) and presents them to the user. Thus, the API module acts as an interface for all data.

3.2.4 Archive module

A major goal of CYBEX-P is the correlation of heterogeneous threat data. The archive module (6.1, 7 in Fig. 3.2) resides in the archive cluster and consists primarily of a set of parsing scripts. As mentioned earlier, the cache data lake (5) is encrypted with the public key of the archive server (6.1). The archive server – (1) gets the encrypted data from the cache data lake (2) decrypts the data using its own private key (3) parses the data into TAHOE, and (4) stores the data in the archive DB (7).

3.2.4.1 Performance Challenge

The archive module potentially handles hundreds of different data formats from thousands of sources. It is reconfigured every time CYBEX-P connects to a new data source. Moreover, it checks each piece of new data against the entire database to determine if it's a duplicate (explained in subsubsection 4.1.6.2).

Although data parsing is a trivial task, we have the made archive module separate because the archive module potentially handles hundreds of different data formats from thousands of different sources.

Moreover, CYBEX-P checks each piece of data for a duplicate against all other data in the database. For example, if CYBEX-P sees an IP in a piece of new data, it checks if the IP exists in another piece of data. The reason for such a design is explained in detail in section 4.1.

However, this design choice, along with the data volume and variety, poses a significant performance challenge for the archive cluster.

3.2.4.2 Design Choices

To tackle this, we have made the archive module separate. This lets us optimize the software or scale the hardware of the archive module without affecting the other modules.

The second design choice that makes the archive module robust actually reflects a key design philosophy of CYBEX-P. The recurrent theme in Fig. 3.2 is placing a database between two software modules. This allows us to modify the data in small atomic transactions. In this case, we have the cache data lake and the archive DB on either side of the archive module.

Since the archive module does not share data with other modules over the system memory, we do not risk losing any data by shutting it down. So, we can shut down the archive module, or any other module for that matter, for maintenance without hampering the overall system.

Finally, we have employed parallel computing to archive the numerous pieces of heterogeneous data. This is a significant contribution to the archive cluster design.

3.2.5 Analytics module

The analytics module (6.3, 7 in Fig. 3.2) works on the archived data to transform, enrich, analyze or correlate them. It has various sub-modules, some of which are described here.

3.2.5.1 Filter sub-module

An analytics filter parses a specific event from raw user data. Multiple filters can act on the same raw data and vice-versa. For example, one filter can extract a *file*

download event from a piece of data while another filter can extract a *DNS query event* from the same data. Filters are discussed in detail in subsection 3.3.5.

3.2.5.2 Enrich sub-module

A particular enrich sub-module can enrich an attribute with related data. For example, we can enrich an URL with the host address. As before, multiple enrichment can be done on the same piece of data.

3.2.5.3 Malicious Scoring sub-module

This is a specialized sub-module that assigns a malicious score to each piece of data and periodically updates the scores. The novel scoring scheme is discussed in section 4.2.

3.2.5.4 Automated Defensive Rule Generation sub-module

This sub-module automatically generates defensive rules (e.g. firewall or intrusion detection system rules) based on the malicious score of the attributes. The rules are published as a feed for users to subscribe. The rules are published as one feed per device model. Users can subscribe according to the devices they own.

3.2.5.5 Phishing URL Detection sub-module

This is another specialized sub-module that automatically detects phishing URLs. We have trained a machine learning classifier with features of many labeled URLs. This sub-module is further described in chapter 5.

3.2.6 Report Module

CYBEX-P is unique in storing cyberthreat data as graphs where the vertices are attributes (e.g. an IP) or events (e.g. an email). This allows CYBEX-P to correlate the data and generate insightful reports. Here, we briefly introduce the report module (❹, ❺, ❻.❷, ❼, ❽, ❾, ❿ in Fig. 3.2).

Users request reports via the frontend client (❿,❾). The API (❹) stores the requests in the cache data lake (❺). The report server (❻.❷) handles those requests by getting relevant data from the archive DB (❼) and aggregating them into reports. It then stores the reports in the report DB (❽). Users can access the reports on demand.

The incident investigation tool, described in subsubsection 3.2.1.1, is also part of the report module. It provides an interactive graph to explore relationships between different attributes and events. To investigate any attribute or event, one has to log in to our investigation tool (3.2.1.1) via any web browser. This tool shows relationships between different attributes and events as an interactive graph. Investigators primarily interact with CYBEX-P through this tool.

Some reports are real-time and can be created upon user request. Some others, however, take up to a couple of minutes to generate. So, we store the reports in the report database (8) and serve them on user demand. The report module also publishes feeds of automatically generated firewall rules. All reports are served via the API server (4).

3.3 Data Flow through Entire Lifecycle

Having discussed the different components and modules of CYBEX-P, in this section we move onto the flow of threat data through those modules. Fig. 3.1 shows the four major functions of CYBEX-P: (1) Data collection, (2) Data analysis, (3) Privacy Preservation, and (4) Reporting. This section demonstrates how CYBEX-P achieves these functionalities by following the entire flow of cyberthreat data through it.

3.3.1 Data Input

A user interacts with CYBEX-P using the frontend webapp client. Users can manually upload threat data, like a spam email, through the frontend webapp (10,9). Users can also configure the collector (3) to automatically collect data from machines like firewalls. Manually uploaded data are directly posted to the API (4) whereas automatically-collected data are handled by the collector.

3.3.2 Privacy Configuration of Data

The frontend further allows the user to attach an ACL to each piece of data. The ACL dictates which attributes are encrypted. The encryption is done at the connector (2) or the webapp client (10) both of which are at the client premises.

Although CYBEX-P cannot access the encrypted attributes, it can still correlate them to generate reports. Users can also share the encryption keys with trusted people. This thesis does not discuss this privacy preservation mechanism. However, it is mentioned here briefly so that the readers get a better understanding of CYBEX-P and can follow through accordingly.

3.3.3 Data Collection

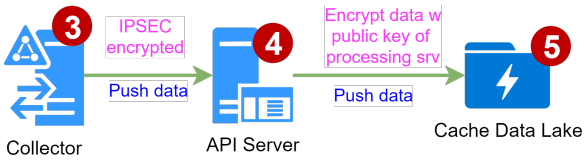


FIGURE 3.4: Data Collection in CYBEX-P.

Automatically collected data is forwarded to the collector (3) over an encrypted channel (TLS). Afterward, it posts the data to the API (4). The API encrypts the data with the public key of the archive cluster (6.1) and stores the encrypted data in the cache data lake (5).

The cache data lake acts as a queue or buffer for all incoming data. It also increases security by removing the need for the API and the archive cluster to communicate directly.

We encrypt the cache data lake with the public key of the archive cluster as the cache data lake is in the demilitarized zone (DMZ). So, the data remains secure even if all the servers in DMZ get compromised.

3.3.4 Data Archiving

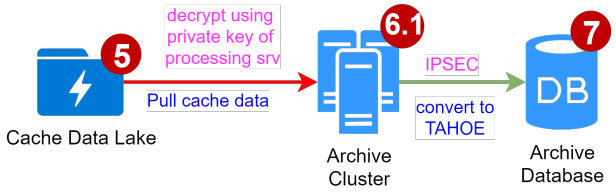


FIGURE 3.5: Data Archiving in CYBEX-P.

The archive cluster (6.1), then pulls the data from the cache data lake (5), decrypts the data using its private key, converts them to TAHOE format and stores them in the archive database (7). All further analyses are performed on TAHOE data. TAHOE is discussed in detail in 4.1.

3.3.5 Data Analytics

The analytics cluster (6.3) transforms, analyzes and correlates data. It achieves that by reading data from the archive database (7), processing the data in the analytics cluster (6.3) and writing the processed the data back in the archive database.

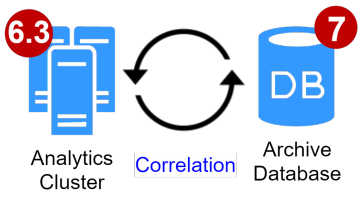


FIGURE 3.6: Data Analytics in CYBEX-P.

This is a continuous process as highlighted by a pair of circular arrows between 6.3 and 7 in Fig. 3.6. It also is the basis for data correlation in CYBEX-P.

For example, consider Fig. 3.7 where 3 filters $F1, F2, F3$ act on a data $D0$ to produce $D1, D2, D3$. An example of such filtering is extracting the source IP, destination IP, and destination port from a firewall log.

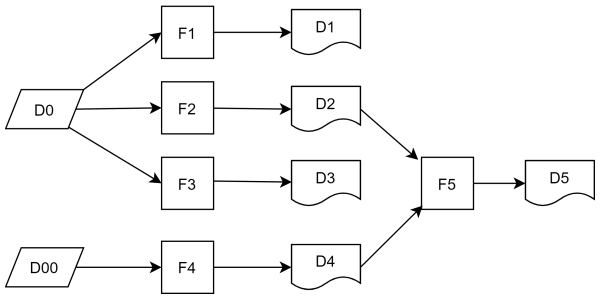


FIGURE 3.7: Data Filtering as a continuous process.

Now, $D2$ is further filtered by $F5$ to create $D5$. On the other hand, $D00$ passes through $F4$ and $F5$ to produce the same attribute $D5$. As a direct consequence of how TAHOE works, $D0$ and $D00$ are now connected to each other in a graph via $D5$. This is a very powerful notion in TAHOE, and we use this to assign malicious scores to new events in 4.2.

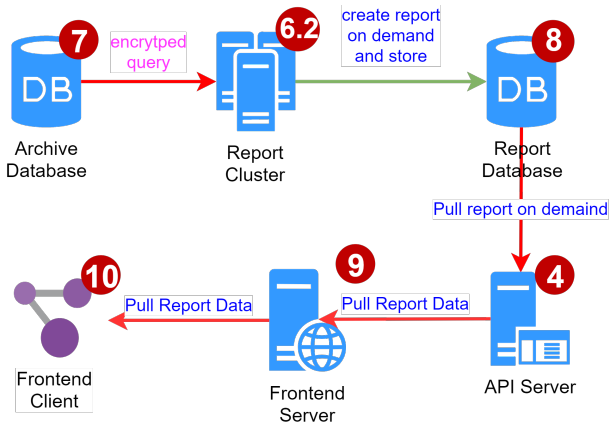


FIGURE 3.8: Data Reporting in CYBEX-P.

3.3.6 Data Reporting

The data pipeline for requesting a report to CYBEX-P is: User ⇒ Frontend client (10) ⇒ frontend server (9) ⇒ API (4) ⇒ Cache data lake (5) ⇒ Report cluster (6.2).

A report can be as simple as counting the occurrence of a particular IP address within a specific time range; on the other hand, a report can be as complex as analyzing the attributes of an URL to determine if it is malicious or benign. Nevertheless, the report cluster stores all the reports in the report database. The user can access the reports as follows: User ⇒ frontend (9, 10) ⇒ API (4) ⇒ Report database (8).

Note that, the overall process of requesting and getting a report is asynchronous.

Chapter 4

TAHOE – A Cyberthreat Language

Now that we have introduced CYBEX-P, in this chapter, we introduce TAHOE the CTL used by CYBEX-P. As discussed in chapter 1, any CIS platform like CYBEX-P potentially handles hundreds of different data formats. Thus, it needs a standard data format and structure to represent threat data. A cyberthreat language (CTL) is a specification of how to format and serialize any kind of threat data. CYBEX-P uses TAHOE instead of other CTLs like STIX[35] or MISP core format [36].

Presently, the most popular CTL is Structured Threat Information Expression (STIX) [35] developed by the MITRE Corporation. While STIX is suitable for mutual data sharing, in subsection 4.1.4 we discuss how it is unscalable for any kind of data analysis. We further explain how STIX-based systems are prone to store duplicate

data. On the other hand of the spectrum, we have MISP core format another very popular CTL. While database queries are fast for MISP, the open data structure of MISP core format defeats that purpose, because different users structure the same data in different ways. Moreover, MISP structures data in only two levels (Events and Attributes); so representing complex data in MISP format is non-intuitive. We discuss the limitations of MISP in subsection 4.1.3.

In contrast to traditional CTLs TAHOE is built from the ground up for faster data correlation while retaining the complexity of machine data. It automatically correlates attributes of incoming threat events with older ones. Finally, lays the foundation for ThreatRank - a novel graph-based algorithm to detect previously unseen malicious events using correlation. In this chapter, we introduce TAHOE as a better alternative to other CTLs and directly compare TAHOE with STIX and MISP. CYBEX-P uses TAHOE for data storage, sharing, and analysis. The complete TAHOE specification is available on GitHub [67].

4.1 Overview of TAHOE

This section outlines a brief overview of how TAHOE structures threat data and also presents the benefits of adopting a graph-based structure with empirical evidence.

4.1.1 TAHOE Data Instance

A piece of TAHOE data is called an `instance` and there are 5 types of TAHOE `instances` —

1. **Raw** A `raw` data instance stores unprocessed user data.
2. **Attribute** The most basic datatype that holds a single piece of information, like an IP address. Fig. 4.1a shows an email address `attribute` .
3. **Object** Groups several `attributes` together, e.g., a file `object` may have a filename and a file-size `attribute` . Fig. 4.1f shows a TAHOE `object` with two `attributes` .
4. **Event** An `event` consists of one or more `attributes` or `objects` along with a `timestamp` . `Events` structure `attributes` or `objects` into complete threat data. Fig. 4.1h shows and email `event` .
5. **Session** A `session` groups arbitrarily related `events` (e.g. events when a user visits a website).

4.1.2 Data Structured as Graphs

TAHOE structures data as graphs, where each TAHOE instance is a graph node. Fig. 4.1 shows an email structured in TAHOE format. Fig. 4.2a visualizes the email as a TAHOE graph with 8 nodes and 11 edges.

```
{"itype": "attribute",
"sub_type": "email_addr",
"data": "jdoe@example.com",
"_hash": "5f07..."}
```


(a) Attribute 'from-email-addr'


```
{"itype": "attribute",
"sub_type": "name",
"data": "John Doe",
"_hash": "22af..."}
```


(b) Attribute 'from-name'


```
{"itype": "attribute",
"sub_type": "email_addr",
"data": "mary@example.com",
"_hash": "b591..."}
```


(c) Attribute 'to-email-addr'.


```
{"itype": "attribute",
"sub_type": "name",
"data": "Mary Smith",
"_hash": "4c90..."}
```


(d) Attribute 'to-name'.


```
{"itype": "attribute",
"sub_type": "subject",
"data": "Saying Hello",
"_hash": "50f2..."}
```


(e) Attribute 'email-subject'.


```
{"itype": "object",
"sub_type": "from",
"_cref": ["5f07..","22af.."],
"_ref": ["5f07..","22af.."],
"_hash": "d722..."}
```


(f) Object 'from'.


```
{"itype": "object",
"sub_type": "to",
"_cref": ["4c90..","b591.."],
"_ref": ["4c90..","b591.."],
"_hash": "da09..."}
```


(g) Object 'to'.

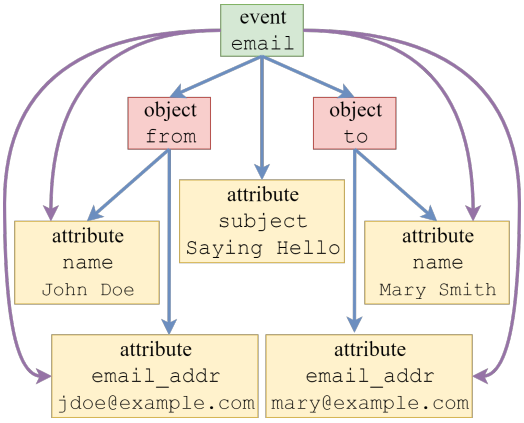

```
{"itype": "event",
"sub_type": "email",
"orgid": "test_org",
"timestamp": 880127706.0
"_cref": ["50f2..","da09..","d722.."],
"_ref": ["4c90..","d722..","5f07..",
    "22af..","da09..","b591..","50f2.."],
"_hash": "f70b..."}
```


(h) Event 'email'.

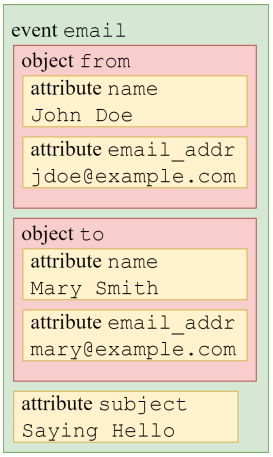
FIGURE 4.1: An email event in TAHOE Format. The complete representation of the email consists of 8 JSON documents. Each document is a node in the TAHOE graph. `_hash` is the globally reproducible and unique ID of a document. `_cref` stores the edges from objects and events to their children. `_ref` stores the edges to both children and grandchildren and so on. The graph is visualized in Fig. 4.2a.

As seen from Fig. 4.1, attributes store actual value or data. Objects and events, on the other hand, do not store any actual data. So, the complete representation of the object in Fig. 4.1f must include the attributes in figures 4.1a and 4.1b. Similarly, the complete representation of the email event in Fig. 4.1h must also include the other 7 documents.

As shown in Fig. 4.1 all TAHOE documents have a field called `"_hash"` . The value of this field is derived from the SHA256 hash of the document and serves as a unique identifier for this document. For example, the string '`attributesubject"Saying Hello"` ' is a unique representation of the subject attribute in Fig. 4.1e. We can calculate the SHA256 checksum of this string to be '`50f2...` '. This checksum is the



(A) The email event as a graph with 8 nodes and 11 edges.



(B) The email event as a nested document.

```
"email": {
  "from": [{
     "email_addr": [
        "jdoe@example.com"],
     "name": ["John Doe"]
     }],
  "to": [{
     "email_addr": [
        "mary@example.com"],
     "name": ["Mary Smith"]
     }],
  "subject": [
     "Saying Hello"]
}
```

(C) The email event as a nested JSON document.

FIGURE 4.2: The email event (green) from Fig. 4.1 as a TAHOE graph and a nested document of objects (red) and attributes (yellow).

unique identifier of the attribute `subject="Saying Hello"` in any TAHOE database.

Ojects and events have an array field called `"_cref"`. The `"_cref"` field stores graph edges. For example, the `"object-from"` in Fig. 4.1f has `"_cref"= ["5f07..","22af.."]`. This indicates, the object is connected to the attributes in Fig. 4.1a and in Fig. 4.1b. Similarly, the event in Fig. 4.1h is connected to two objects in Figures 4.1f and 4.1g and also the attribute in Fig. 4.1e. Fig. 4.2a shows these edges as blue arrows.

The array `"_ref"` stores graph edges to all subsequent nodes including children and grandchildren. This field is used for making queries faster and explained in subsection 4.1.4. Both the blue and purple edges from Fig. 4.2a are stored in `"_ref"`. Note, how the event contains complete information about the email just by referring to the other objects and attributes.

Benefits of this graphical structure are justified in subsection 4.1.5. Also, note that

we draw the edges as arrows because of how edge data is stored in `_ref` . However, in a TAHOE database, a graph can be traversed from both ends, as explained in subsection 4.1.6.6. Objects can also refer to other objects if required.

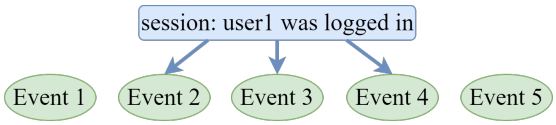


FIGURE 4.3: Events grouped by arbitrary session parameter.

Finally, A TAHOE session is an arbitrary grouping of related events. This allows us to group events based on any condition the user desires. The session in Fig. 4.3 groups 3 events, recorded while *user*1 was logged in.

Events Viewed as Nested Documents Although, TAHOE structures events as graphs, they can be viewed as nested documents. Fig. 4.2b shows the email event from Fig. 4.1 as a nested document. Furthermore, Fig. 4.2c shows the JSON representation of the nested document. This JSON document is obtained by traversing the graph starting from the event node. The graph edges are stored in the `_cref` arrays. Analysts can choose to view an event as a document or as a graph depending on their need. For all kinds of machine analysis (e.g query), however, the graphical structure of Fig. 4.2a is more suitable.

event email
email-src-display-name
= John Doe
email-src = jdoe@example.com
email-src-display-name
= Mary Smith
email-dst = mary@example.com
email-subject = Saying Hello

(A) The email in Fig. 4.1 as MISP format. MISP struggles to structure complex data and specifies non-intuitive `attribute` types.

```
{
"0": {
    "type": "email-addr",
    "value": "jdoe@example.com",
    "display_name": "John Doe"
    },
"1": {
    "type": "email-addr",
    "value": "mary@example.com",
    "display_name": "Mary Smith"
    },
"2": {
    "type": "email-message",
    "from_ref": "0",
    "to_refs": ["1"],
    "date": "1997-11-21T15:55:06Z",
    "subject": "Saying Hello"
    }
}
```

(B) The email in Fig. 4.1 as STIX format.

```
{
"0": {
    "type": "ipv4-addr",
    "value": "1.2.3.4",
    "belongs_to_refs": ["3"]
    },
"1": {
    "type": "ipv4-addr",
    "value": "2.3.4.5"
    },
"2": {
    "type": "network-traffic",
    "src_ref": "0",
    "dst_ref": "1",
    }
"3": {
    "type": "as"
    "number": 42
    }
}
```

(C) A network packet in STIX format.

FIGURE 4.4: The email event from figures 4.1 and 4.2 in MISP and STIX format and a network packet in STIX format.

4.1.3 Representing Complex Data – TAHOE vs. MISP

Traditional CTLs like MISP often struggle to represent complex data. For example, Fig. 4.4a shows the email event from Fig. 4.1 in MISP [36] format.

The key problem is, `email-src` and `email-dst` are two different attribute-types. So, to fetch all emails to and from `jdoe@example.com` , one has to perform 2 queries – `email-src = jdoe@example.com` and `email-dst = jdoe@example.com` . Moreover, MISP represents a lot of information in the attribute-type. So MISP data structure includes cumbersome attribute-types like `passenger-name-record-locator-number` or non-intuitive attribute-types like `filename|md5, filename|sha224, filename|sha256` etc.

Firstly, TAHOE can store arbitrarily complex data because TAHOE objects can be infinitely nested and can refer to other objects. As seen if Fig. 4.1, TAHOE has simple attribute types like, `email_addr` or `subject` . Secondly, a TAHOE event is connected to all of its attributes via the `_ref` array. So, to fetch all emails to and from `jdoe@example.com` , TAHOE requires only 1 query – `email_addr = jdoe@example.com` .

4.1.4 Indexing & Scalability – TAHOE vs. STIX

Earlier in chapter 2, we claimed that "While STIX is perfect for mutual data sharing, it is unscalable for any kind of data analysis." This subsection provides a detailed explanation in support of our claim. Since, both STIX and TAHOE use JSON documents we will have to store them in a NoSQL database. For this discussion, we consider the most popular NoSQL database - MongoDB [68].

Consider, the STIX document in Fig. 4.4b. It has 3 keys - `"0"` , `"1"` , `"2"` . Here, `"0"` and `"1` are JSON objects with 3 keys each whereas `"2"` is a JSON object with 5 keys. Assume, we want to *fetch all emails from jdoe@example.com.* The MongoDB syntax for that query is, `find({"0.value":"jdoe@example.com"})` .[1] This query will get all JSON documents in the database which have `"0.value"="jdoe@example.com"`

[1]The actual query is `find({"0.type":"email-addr", "2.type":"email-message", "0.value": "jdoe@example.com"})`. We have shortened the queries in the example for clarity.

. Similarly, if we want to *fetch all emails with subject="Saying Hello"* the query is `find({"2.subject": "Saying Hello"})` .

Now, these queries will take forever in a decent sized database unless we index the keys `"0.value"` and `"2.subject"` . Similarly, in Fig. 4.4c, to efficiently lookup all the network traffic events of `"type"="as"` , we must index the `"3.type"` key. Eventually, for these two event types, we must index a total of 16 keys – `"0.type", "0.value", "0.display_name", "1.type", "1.value", "1.display_name", "2.type", "2.from_ref", "2.to_refs", "2.date", "2.subject", "0.belongs_to_refs", "2.src_ref", "2.dst_ref", "3.type", "3.number"` . Indexing in this manner creates 3 problems for us:

1. Not all documents have all the keys. For instance, the network traffic event in Fig. 4.4c does not have the `"2.subject"` key. So, the indexing will be inefficient.

2. MongoDB only allows 64 keys to be indexed in a database collection. As discussed earlier, we have 16 keys only for two types of events. As we encounter, more event types with arbitrary structures, we will have hundreds of keys that need indexing. Indexing so many keys is not feasible.

3. Some, fields have large values. For example, RFC2322 states that the email subject has no length restrictions. Which means the `"2.subject"` field in Fig. 4.4b can be larger than 1024 bytes. However, MongoDB cannot index a field larger than 1024 bytes. So, large fields in a STIX document can never be indexed.

Going back to our earlier discussion queries like `find({"0.value": "jdoe@example.com"})` will be too slow in a database if the fields are not indexed. That is why we claim that STIX is unscalable for data analytics.

TAHOE incorporates a novel solution to these challenges. Consider the query *fetch all emails with subject="Saying Hello"*. This is a two-step process in TAHOE –

1. We create the string '`attributesubject"Saying Hello"` '. This string is an unique representation of this subject attribute. We then calculate the SHA256 checksum of this string to be '`50f2...` '. This checksum is the unique identifier of the attribute `subject="Saying Hello"` in any TAHOE database.

2. In the second step, we perform the following MongoDB query - `find({"_ref": "50f2..."})` . [2] [3]

These two steps will return all the emails which have `subject="Saying Hello"` . In terms of TAHOE graph, we are essentially querying all events which are connected to the TAHOE attribute `subject="Saying Hello"` .

Now, in TAHOE, we only query one key - `"_ref"` ; so, we only index this one key. Therefore, we will never pass the 64 keys limit of MongoDB. Also, all events have the `"_ref"` field, so the indexing will be efficient. Finally, each element in the `"_ref"` array is a SHA256 checksum, which means each of them is 256 bits long. So, we will

[2] The actual query is `find({"itype":"event", "sub_type":"email", "_ref": "50f2..."})`.

[3] We have made a TAHOE backend library that interfaces with MongoDB to automate the whole query process. So, users will not actually have to calculate the checksums and then query the MongoDB. The library is available at https://github.com/cybex-p/tahoe

not violate the 1024 bytes limit on indexed fields. Thus TAHOE takes care of all the 3 problems mentioned earlier for STIX.

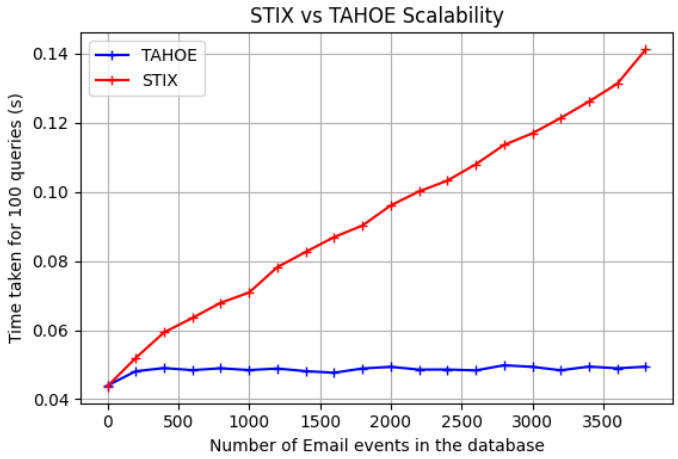
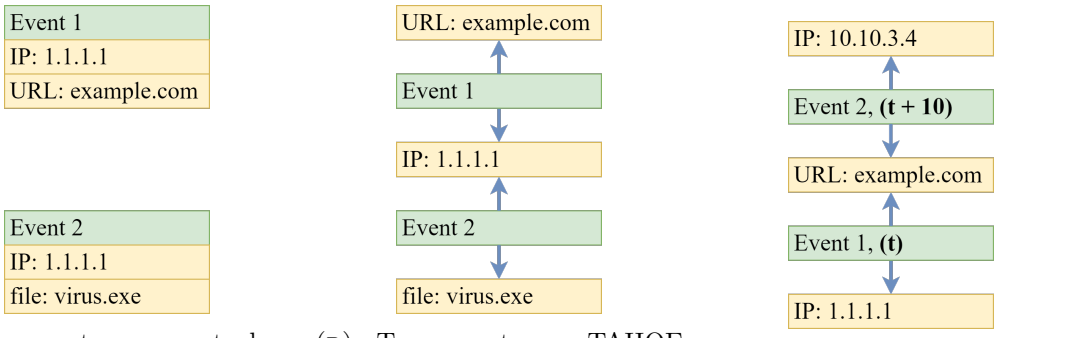


FIGURE 4.5: Scalability of STIX vs TAHOE. The x-axis is the total number of email events in the database. The y-axis is the total time taken for 1000 queries in the databases. The query for the STIX database is `find({"0.value": "jdoe@example.com"})`. The query for the TAHOE database is `find({"_ref": "50f2..."})`.

Fig. 4.5 compares the scalability STIX 2.0 and TAHOE. The graph compares the time take for the `find({"0.value": "jdoe@example.com"})` query for the STIX database with the `find({"_ref": "50f2..."})` query for the TAHOE database. The x-axis is the total number of email events in the database. The y-axis is the total time taken for 100 queries in the databases. The plot clearly shows that the time-required grows linearly with the database size for STIX. This is because the key `"0.value"` is not indexed and every query becomes a linear search in the database. The time required for the TAHOE database, however, stays constant as the number of emails grows in the database. This is because the key `"_ref"` is indexed in the database. And since

indexed keys are stored as hash tables in the RAM, every lookup takes roughly the same time despite the size of the database.

4.1.5 Intrinsic Correlation of Graphical Data



(A) Two events as separate documents. (B) Two events as TAHOE graph. (C) A time-varying attribute.

FIGURE 4.6: Fig. 4.6b shows how TAHOE intrinsically correlates the two separate events from Fig. 4.6a based on their common attribute. Fig. 4.6c shows an example TAHOE graph for a time-varying attribute.

Traditional CTLs, store threat data as separate documents like shown in Fig. 4.6a. These data are difficult to analyze because the events lack any direct correlation with their attributes. TAHOE, on the other hand, represents data as graphs like in Fig. 4.6b. Here, two separate events are automatically connected by their common `attribute` (`1.1.1.1`) in TAHOE. Such 'intrinsic correlation' is a powerful feature of TAHOE, because if someone looks up `example.com` she will immediately see that `virus.exe` is related to it. This is a major strength of our investigation tool (subsubsection 3.2.1.1). Moreover, we leverage this feature to formulate a novel malicious event detection mechanism in section 4.2.

Furthermore, TAHOE can correlate time-varying attributes. As shown in Fig. 4.6c, lets assume the Tahoe database records two events - Event 1 at time **t** and Event 2 at time $\mathbf{t} + \mathbf{10}$. Also assume, During event 1, at time **t**, the domain `example.com` resolves to `1.1.1.1` and during event 2, at time $\mathbf{t} + \mathbf{10}$, the domain `example.com` resolves to `10.10.3.4` . Now, the TAHOE database will connect both the IP addresses to the domain in a graph like that shown in Fig. 4.6c.

Please note that, both IP addresses are associated with `example.com` after Event 2 is recorded. Also note that, the IPs are not directly connected to `example.com` , rather connected via their respective events. For example, `1.1.1.1` is connected to `example.com` via Event 1 (time **t**) and `10.10.3.4` is connected to `example.com` via Event 2 (time $\mathbf{t} + \mathbf{10}$).

So, if someone queries `example.com` , they will be able to see the complete graph in Fig. 4.6c including the timestamps. From, the graph any person or machine can deduce that `example.com` was associated with `1.1.1.1` at time **t** and was associated with `10.10.3.4` at time $\mathbf{t} + \mathbf{10}$. Thus, the time aspect of the relationship is preserved.

Moreover, users can specify time ranges when querying the TAHOE database to filter out one of these events. For example, if any user or machine queries events related to `example.com` between $\mathbf{t} + \mathbf{5}$ and $\mathbf{t} + \mathbf{15}$, they will only see Event 2 in the graph.

4.1.6 Features of TAHOE

4.1.6.1 Data Normalization

TAHOE normalizes different formats of same type of data. Consider two firewalls from two different vendors. Their log data will be formatted differently despite having the same type of data. TAHOE normalizes such differences by converting them into the same structure.

4.1.6.2 Data De-duplication

TAHOE prohibits duplicate data. For example, there can only be one instance of the IP 1.1.1.1 in a TAHOE database. This saves CYBEX-P a lot of storage by not storing the same IP in different `events` . TAHOE achieves this de-duplication of data by creating a globally reproducible hash of the data.

4.1.6.3 Database Independence

Although TAHOE is a graph-based CTL we did not use a graph database as a container for TAHOE. In other words, all the information, including the edge data, of a TAHOE graph is stored in the JSON documents of the TAHOE `instances` , as shown in Fig. 4.1.

Furthermore, as described in 4.1.7 we have developed a universal threat data query language (TDQL) to communicate with any TAHOE storage. These two contributions make TAHOE a database-independent CTL.

4.1.6.4 Optimized for Indexing

Subsection 4.1.4 discusses how TAHOE is optimized for indexing in databases and compares the query performance of a STIX database with that of a TAHOE database. The ability to query related data is a novel feature of TAHOE and enables CYBEX-P to perform advanced analytics on TAHOE data.

4.1.6.5 Globally Unique & Reproducible Data for Conflict-free Sharing

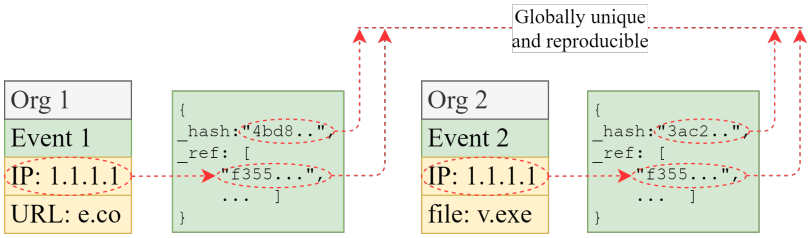


FIGURE 4.7: TAHOE id and edges are globally unique and reproducible, making them collision free.

TAHOE data are globally unique and reproducible. As shown, in Fig. 4.7, the IP `1.1.1.1` has the same unique id (its hash) in two different organizations. Consider, `Org 1` shares `Event 1` with `Org 2`. If `Org 2` had a different id for `1.1.1.1` it would have to update the `_ref` array of `Event 1`. But, as hashes are reproducible yet unique, this is not required. Note that, event hashes include a timestamp (not shown

in figure). Hence, two separate events will have different hashes even if they have the same attributes.

4.1.6.6 Bidirectional Edges for Versatile Queries

TAHOE edges are bidirectional. As seen in Fig. 4.7, edge data is stored in the `event` only. This is because an IP like 8.8.8.8 (public DNS) can potentially get connected to millions of `events` . If we store the hash of all these `events` in the IP `attribute` , it would result in an unbounded growth of its edge array. So, we store the edge info in the `events` . However, it takes only one pass over the database, to get all `events` that have a particular hash in their edge array. So, the edges are bidirectional for all intents and purposes.

4.1.7 Threat Data Query Language (TDQL)

TAHOE aims to standardize the structuring of threat data in terms of `attributes, objects, events` and `sessions` . This would allow users to query threat data using those terms. An example query could be `fetch all events which include the attribute 1.1.1.1` . At present this is not possible because `event` or `attribute` are not standardized terms for any existing database. For example, if a person queries an SQL database for `events` it would not know what to return, because `event` is not a standard term for SQL.

To that end, we have developed a universal threat data query language (TDQL) for TAHOE. TDQL acts as a layer between a database and a user. Additionally, TDQL is tailor-made for threat data and addresses their nuances. While SQL depends on the structure of database tables, TDQL speaks in terms of `attributes`, `objects`, `events` etc. So, irrespective of the data storage or delivery protocol, a user can always fetch any threat data from any database. Additionally, having a dedicated TDQL makes TAHOE, database-independent. However, detailed documentation of TDQL is beyond the scope of this research work.

In summary, CYBEX-P uses TAHOE which stores data as graphs. TAHOE leverages existing graph algorithms and basic graph operations to generate deep insights into seemingly unrelated cyberthreat data. To the best of our knowledge, such correlations have not been studied in the cybersecurity industry making CYBEX-P and TAHOE a novel endeavor.

4.2 ThreatRank to Detect Malicious Events

Earlier in subsection 4.1.5 we introduced how TAHOE intrinsically correlates data. Here, we extend upon it by formulating an algorithm, called ThreatRank, to assign a malicious score to each `event` in a TAHOE database. The score essentially sorts the `events` from most malicious to least malicious. In 4.3.5 we justify this algorithm with real data.

It helps out the security administrator because it is impossible to manually analyze all `events` (e.g. all firewall logs) in a regular network. Because of the assigned score, the security administrator can start from the `event` that is most likely to be malicious. In this section, we discuss how we assign the score to each `event`.

4.2.1 Attributes in a Malicious Context

While a malicious `event` (e.g. a spam email) stays malicious for eternity, the same is not true for `attributes`. For example, a website can be hacked and used to distribute malware for a week; after which it is restored by the website admin. Here, the website `URL attribute` is malicious for a week and becomes benign afterward.

Similarly, not all `attributes` of a malicious `event` are malicious. For example, if X receives a spam email from Y, only Y is a malicious `attribute` not X. X is the victim and a benign `attribute` in this `event`.

For these reasons, TAHOE never classifies an `attribute` (e.g. an IP) as malicious. Rather TAHOE maintains a special edge, called a `_mal_ref` between an `event` and an `attribute`. For example, if an IP `1.1.1.1` is seen in a malicious context in a firewall log `event`, the IP is connected by a `_mal_ref` with the log `event`.

As a result, TAHOE can count the number of times a particular `attribute` has been seen in a malicious context vs in a benign context. Furthermore, we utilize this notion to formulate the ThreatRank algorithm below.

4.2.2 ThreatRank Algorithm

Consider, $\mathbb{A} = \{a_1, a_2, ..., a_m\}$ is the set of all `attributes` and $\mathbb{E} = e_1, e_2, ..., e_n$ is the set of all `events`. $\mathbb{E}_{mal} \subseteq \mathbb{E}$ is the set of known malicious `events`. We define $\mathbb{I}_{mal} = \{k \mid e_k \in \mathbb{E}_{mal}\}$. We want to determine the ThreatRank (TR) of a new `event` e_p.

We define $w_{i,j} = \{e_i, ..., a_x, e_y, a_z..., e_p\}$ as the j^{th} path from e_i to e_p. Note that, the path encounters `attributes` and `events` in an alternating fashion and has distinct nodes.

Then the contribution of $w_{i,j}$ to the ThreatRank of e_p is calculated using the recurrence equation—

$$TR_{w_{i,j}}[k] = 0.998^{d_{k-1}} \times \frac{TR_{w_{i,j}}[k-1]}{L(w_{i,j}[k-1])} \tag{4.1}$$

where, $TR_{w_{i,j}}[1] = -1$; $d_k = 0$ for an `attribute` and for an `event`, d_k is the number of days passed since the `event` e_k was recorded; $L(x)$ is the degree of node x.

Assume, there are t_i paths from e_i to e_p. We define the set $\mathbb{W} = \{w_{i,j} \mid i \in \mathbb{I}_{mal}; j \in [1, t_i]\}$. $\mathbb{W}$ basically includes all the paths from all known malicious `events` to the new `event`. The total ThreatRank of e_p is then calculated as —

$$TR(e_p) = \sum_{w \in \mathbb{W}} TR_w[t_i] \tag{4.2}$$

Algorithm 1 lists the pseudocode for ThreatRank. The code is written using TAHOE terminology.

4.2.3 Why 0.998?

We multiply the ThreatRank of each `event` by 0.998^{d_k}. Here, d_k is the number of days passed since the `event` e_k was recorded. The value 0.998 is chosen such that after 1 year an `event` is half as significant ($0.998^{365} = 0.48$) as a recent `event` ($0.998^0 = 1$). The same `event` is only one-fourth as significant ($0.998^{730} = 0.23$) after two years.

4.2.4 Who Classifies Malicious Events & Edges?

Malicious `events` or edges can be classified in three ways — (1) manually by CYBEX-P admin after analysis (2) by user voting (3) automatically for some data. For example, an IP that tries to connect to a honeypot is automatically classified as a malicious IP in this context.

4.2.5 ThreatRank vs Degree Distribution

The malicious score of an `event` can be calculated as the number of malicious edges connected to it; which is equivalent to its degree in the graph. However, an `attribute` that is seen in malicious context in an `event` may show up in benign context in thousands of other `events`. Such, an `attribute` should contribute less to the malicious

Algorithm 1: ThreatRank

Input: $\mathbb{E}$, $\mathbb{E}_{mal}$

```
Function getRelated(node)
    if node.type = "event" then
        return node._ref
    end
    related ← [] ;
    for event in 𝔼 do
        if node in event._ref then
            related.append(node)
        end
    end
    return related ;
end
Function findPaths(src, dest, currentPath)
    if src = dest then
        return currentPath
    end
    related ← getRelated(src) ;
    paths ← [] ;
    for r in related do
        if r in currentPath then
            continue
        end
        paths.append(findPaths(r, dest, currentPath+[r])) ;
    end
    return paths ;
end
Function threatRankPath(path)
    tr ← −1 ;
    for node in path do
        L ← degree(node) ;
        d ← 0 ;
        if node.type = "event" then
            d ← node.daysOld
        end
        tr ← tr × 0.998**d / L ;
    end
    return tr ;
end
```

```
39 Function threatRank(newEvent)
40 |  allPaths ← [],   TR ← 0 ;
41 |  for event in 𝔼_mal do
42 |  |  allPaths.append(findPaths(event, newEvent, []))
43 |  end
44 |  for path in allPaths do
45 |  |  TR ← TR + threatRankPath(path) ;
46 |  end
47 |  return TR ;
48 end
```

score than another `attribute` which is present in malicious context in all `events`. ThreatRank takes this into consideration.

4.3 Implementation and Experimental Evaluation

We have implemented CYBEX-P for experimental evaluation, with 5 data sources, along with 4 instances of MongoDB and 4 servers to house the different modules. We have collected about 314 billion events from several sources. As of now, we have a fully functional prototype of CYBEX-P with the complete data flow. Fig. 4.8 shows the network diagram of our current CYBEX-P implementation.

4.3.1 Sources

The sources we used for our demonstration are:

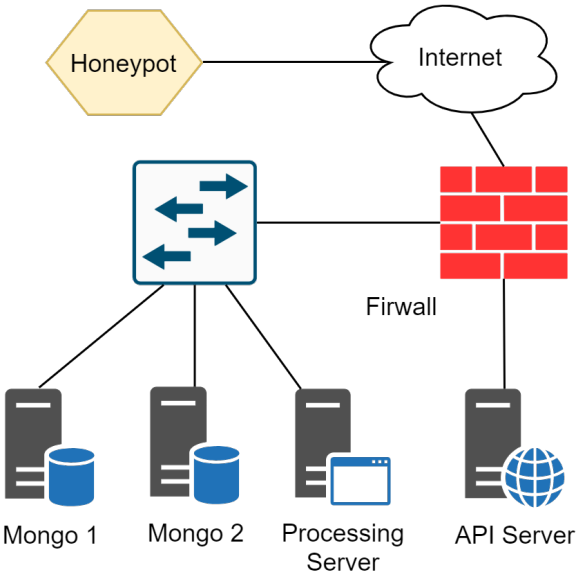


FIGURE 4.8: Network Diagram of CYBEX-P

1. Cyberthreat intelligence (CTI) from University of Nevada, Reno's MISP [24] instance.

2. SSH login attempts collected by four different instances of 'cowrie' honeypot [69].

3. Firewall log data from our honeypot system.

4. Feed of phishing URLS from Phishtank [70].

5. Feed of phishing URLS from OpenPhish [71].

4.3.2 Network Architecture

As seen Fig. 4.8 there are 4 physical servers in the CYBEX-P network. Furthermore, there is a switch and a firewall in the network. We also maintain a family of honeypots

around the world which are not shown in Fig. 4.8. These honeypots are connected to the CYBEX-P premises by public internet.

The API server in our network houses the input module and the API module including the cache data lake. The two database servers Mongo 1 and Mongo 2 store the TAHOE DB. Both of these databases store the same data for higher availability by forming a MongoDB replicaset. The processing server is the brain of CYBEX-P. It does all kinds of processing and data analytics. The archive and the analytics software modules live here. The firewall plays a crucial role in our threat model and security design. It keeps the API server separate from the inside network as discussed in subsection 3.2.3.

4.3.3 Complexity & Scalability

To test the scalability of CYBEX-P, we have fed 300000 lines of *iptables* firewall log into it. Then we have recorded the time taken to process N log messages. We have measured the time taken from input to storing in the archive DB as TAHOE events. A line of best fit is drawn among the data points. The result is shown in figure 4.9.

The test was done with a run-of-the-mill computer, to show the complexity of our mechanism and hence forecast the scaling and growth of the system. For this reason, the processing was done using a single core and the absolute times are of less interest here.

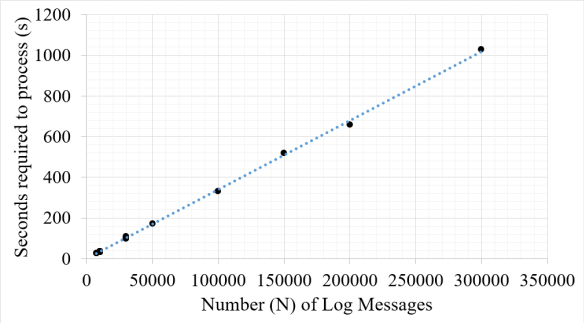


FIGURE 4.9: Evaluation of Complexity

It can be inferred from the test that the overall complexity is linear. Furthermore, each log message can be processed independently of the other. This makes the system suitable for horizontal scaling using distributed computing paradigms because the processing time grows linearly with the input size. This is desirable in a system like this which handles a large volume of data. This makes the system suitable for horizontal scaling using distributed computing systems. Furthermore, each log message is processed independently of each other making the process ideal for distributed computing.

4.3.4 Data Compression by TAHOE

As discussed in 4.1.6.2, TAHOE de-duplicates data, meaning there is only one instance of the IP `1.1.1.1` in our TAHOE database. Furthermore, `events` never store the actual value of an `attribute`, only a reference to it. The reference is the SHA256 hash of the attribute and only takes 32 bytes of storage. So, if an `attribute` is

repeated in another event, TAHOE takes only 32 bytes of extra storage. As a result, TAHOE automatically achieves significant data compression as shown in Fig. 4.10.

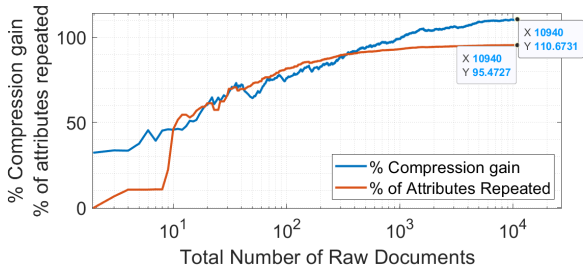


FIGURE 4.10: Data Compression in TAHOE

As seen in Fig. 4.10, initially the compression gain is below 100%. However, as the percentage of repeated attributes grows so does the compression gain. Here, TAHOE achieves a compression gain of 10.7% for only about 11 thousand pieces of raw threat data, collected from our Cowrie honeypots.

4.3.5 ThreatRank Verification by Case Study

In section 4.2 we have formulated an algorithm called ThreatRank (TR) to detect malicious `events`. Here, we verify this algorithm. using the 'Intrusion Kill Chain' [72] dataset from Lockheed Martin.

4.3.5.1 Intrusion Kill Chain and Correlation

Authors of [72] formulated the 7 phases of an intrusion kill chain (also known as cyber kill chain) – (1) Reconnaissance, (2) Weaponization, (3) Delivery, (4) Exploitation,

(5) Installation, (6) Command and Control, (7) Actions. It is desirable to detect an attack in an early phase.

In their case study, there are 3 related intrusion attempts. The first attempt delivered a malicious file via email. Although at first, the email looked benign, it was later flagged as malicious because it had a malicious attachment. Note that, by that time the attack has already passed phase 3 undetected.

The next two intrusion attempts also delivered malicious files via emails. However, both these emails had similarities with the first email. As a result, the defenders could detect the attack even before analyzing the malicious files.

In other words, the defenders detected these two attempts in phase 3, not later, because the emails are correlated. Fig. 4.11 shows the common attributes in these emails as a TAHOE graph.

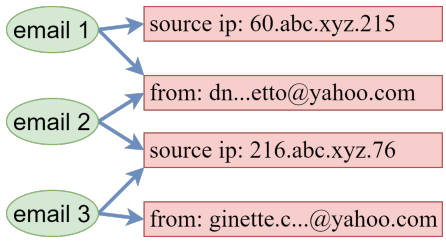


FIGURE 4.11: Three emails from three separate intrusion attempts are intrinsically correlated in TAHOE because of common attributes.

4.3.5.2 Automatic Intrinsic Correlation by TAHOE

While the correlations in Fig. 4.11 are trivial, it is impossible for defenders to manually analyze all emails. TAHOE automates the process by intrinsically correlating

the 3 emails based on their common attributes as shown in Fig. 4.11.

However, correlating them is only half the battle. The correlation must be quantified before alerting the analysts. That is where ThreatRank steps in.

4.3.5.3 ThreatRank to Quantify Correlations

We assume that `email 1` has already been flagged as malicious in the TAHOE database. So, we assign a fixed ThreatRank (TR) of 1 to `email 1` and mark the two edges in Fig. 4.11 as malicious. Then we simulate ThreatRank on the graph to get the results in Table 4.1. We have also added a benign email called `email 4` to the TAHOE database. `email 4` shares no common attribute with any of the `emails 1,2,3`.

TABLE 4.1: ThreatRank of 4 emails calculated on 3 dates

Email	arrival_2	arrival_3	$\text{arrival}_1 + 365$
`email 1`	-1	-1	-1
`email 2`	-0.18	-0.15	-0.07
`email 3`	N/A	-0.08	-0.03
`email 4`	0	0	0

Here, arrival_1 is the date of arrival of `email 1` and $\text{arrival}_1 + 365$ is one year later. Note that, `email 1` has fixed ThreatRank of -1 because it is already analyzed by an analyst. ThreatRank is calculated for unknown events only. Also, in the dataset `email 2` arrives 1 day after `email 1` and `email 3` arrives 20 days after `email 1`.

`email 2` has a TR of -0.18 while `email 3` has a TR of -0.08 on respective arrival day. `email 3` has a lower TR because `email 2` is directly connected to `email 1`, whereas `email 3` is one hop away from `email 1`. Also, as expected their TR becomes almost half after a year. For all the simulations, TR of `email 4` remains 0 because it shares no common attributes with the other emails.

Chapter 5

CYBEX-P for Phishing URL Detection

As described in chapters 1 and 2, existing information sharing platforms have limited or zero support for data analysis. In this paper, we have introduced CYBEX-P as a complete CIS platform to tackle that limitation. Our vision for CYBEX-P is to provide Infrastructure as a Service (IaaS) for all kinds of threat analysis. Now that we have discussed CYBEX-P in detail, we are ready to present a use-case of how CYBEX-P can be used for advanced data analysis. To that end, this chapter presents Phishly - a real-time phishing URL detector [73] using CYBEX-P infrastructure.

The key goal of a phishing attack is collecting valuable information like passwords or credit card numbers from unsuspecting visitors. Phishing is a plague of the internet

that equally affects people and organizations. A report from Ponemon Institute estimates that phishing costs US companies $14.8 million on average in 2021 [74]. This is a striking rise from $3.8 million in 2015. Therefore, it is evident that traditional defense mechanisms are failing to detect phishing URLs.

This calls for an approach to detect new phishing URLs based on the older ones on zero-day. Such a system needs to be automated and real-time. It should automatically analyze the current landscape of phishing URLs and generate a model from those URLs. It should further provide an automated mechanism for humans and machines to check if an URL is benign or malicious. Building a framework like this is a major undertaking faced with many challenges.

Fortunately, CYBEX-P already provides an interface for automating all of these tasks. In the subsequent sections of this chapter, we will discuss how CYBEX-P infrastructure can be used as a service (IaaS) to build this new framework. Providing such a robust interface for all kinds of threat data analysis to future researchers is one of the major achievements of CYBEX-P.

5.1 Introduction

A major type of cyberattack, that affects people and businesses alike, is phishing [75]. The attacker of a phishing attack tries to gather sensitive information of a user by disguising as a trustworthy third party. Such an attack usually consists of

directing users to a fake website that resembles another legitimate website. The URL of the fake, phishing website is typically distributed via emails or instant messages. Oftentimes it is very difficult or impossible for a user to detect such fake websites just by looking at the content of the web page. Phishing attacks cost a mid-sized company about 1.5 million US dollars on average[76].

It should be noted here that phishing URLs are part of a broader set called malicious URLs. Other types of malicious URLs include drive-by-download URLs, spam URLs, etc. Even though the rest of the work discusses detecting phishing URLs only, the generalized procedure can easily be tweaked for detecting all kinds of malicious URLs.

The most popular technique used to detect phishing URLs is the use of blacklists [77]. A blacklist is simply a list of malicious URLs, periodically updated by community users or cybersecurity experts. However, the Webroot Threat Report estimates [78] that nearly 1.5 million phishing websites are created every month. As a result, it is not possible to blacklist all phishing websites and corresponding URLs on time. So, an automated framework to detect new phishing URLs is required. Such a system should detect previously unseen phishing URLs with high accuracy without any human interaction.

Many approaches have been explored towards such an end including machine learning [79–83]. A machine learning approach begins with the collection of a dataset. The dataset includes different features and labels (benign vs. malicious) of a large number of URLs. A classifier is then trained with the collected dataset.

The underlying assumption is that a malicious URL has a significantly different feature distribution than a benign URL. As a result, a good machine learning approach should be able to differentiate between benign and malicious URLs based on those features. This hypothesis has been verified in numerous previous works [77, 84–88] using diverse datasets and state-of-the-art machine learning algorithms.

While many of the previous works show promising results, adoption of such a mechanism in the industry is yet to be seen. This is because it is prohibitively slow to detect malicious URLs using machine learning algorithms in a real-time setup. Moreover, the URL space is highly unbalanced with many more benign URLs than phishing URLs. Furthermore, the URL space is dynamic and changes over time, meaning the classifier must be updated periodically. Additionally, the growth of the URL space is unbounded, which means we cannot use traditional batch learning methods to train on all URLs. Due to these challenges, a complete, standard framework for detecting malicious URLs has not been proposed.

This work aims to tackle these challenges by introducing a complete automated framework for phishing URL detection. While the framework is built for detecting phishing URLs only, the general approach applies to identifying all types of malicious URLs.

The primary contribution of this part of the work is introducing a complete, automated, real-time framework to detect phishing URLs. The framework is completely automated from data collection to analysis to detection. In this work, We have used an online or incremental learning method to tackle the unbounded growth of the URL

space. We further propose a delayed feature collection algorithm and a selective sampling algorithm to increase the performance of our framework. We have tested our system's accuracy and performance with a very diverse and unbiased dataset. Our system can label an URL with 86% accuracy without suffering from any performance bottleneck.

The table 5.1 shows how this system uses CYBEX-P.

TABLE 5.1: How CYBEX-P provides Infrastructure as a Service (IaaS) for the Phishing URL Detection System

Module	**How it is used**
Frontend	(1) Get user request to check an URL
Input	(1) Fetch training URLs from Phishtank using collector
API	(1) To interface with collector (2) To store training URLS in Cache datalake
Archive	(1) To parse PhishTank and user data into TAHOE (2) To store parsed data in Archive DB
Analytics	(1) To generate training model from PhishTank data (2) To store training model in Archive DB (3) To classify input URL using training model
Report	(1) To store result in Report DB (2) To show result to user

5.2 System Architecture

Having introduced the framework, in this section, we dive into the system architecture of the proposed system. Fig. 5.1 shows the system architecture of this system. As

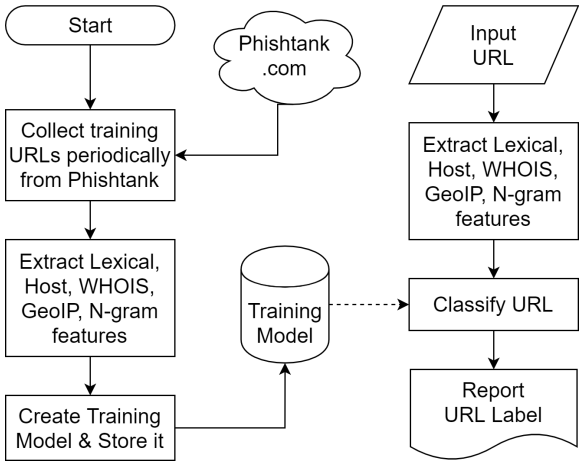


FIGURE 5.1: System Architecture of the Phishing URL Detection system

it uses CYBEX-P's infrastructure, it is explained using CYBEX-P components from Fig. 3.2.

Here, Phishtank.com stores raw URLs (1 in Fig. 3.2). It provides an API (2) to share the URLs. Note that Phishtank's API is the connector from CYBEX-P's perspective. CYBEX-P collector (3) calls that API periodically to get new URLs and post them to the CYBEX-P API (4). CYBEX-P API puts them in the cache datalake (5). The archive cluster (6.1) parses the raw URLs into TAHOE events and stores them in the archive DB (7). The analytics cluster (6.3) enriches the URLs with 5 sets of features. The feature sets are described in 5.4. The analytics cluster also trains a classifier model.

On the other hand, a user can input an URL via the frontend (10, 9) webapp. The analytics cluster classifies this URL, as benign or phishing, using the previously trained model. The report cluster (6.2) creates a report out of the classification label, and stores it in the report DB (8). The report is shown to the user via the frontend.

5.3 Dataset

We have collected about 36,000 phishing URLs from Phishtank. We have also collected approximately 60,000 benign URLs from Phishtank which were previously reported as suspicious but later analyzed to be benign. On the other hand, several previous works [89, 90] used web crawling to generate benign URLs out of highly ranked websites only. This approach often results in a biased dataset. Our benign URL dataset is better aligned with the real world.

5.4 Features

We extract five types of features from each URL: lexical, host, GeoIP, domain WHOIS, and n-gram. Each feature set is described below:

5.4.1 Lexical Features:

Lexical features are based on the URL string itself. Several examples of typical lexical features are the number of characters in the URL, the number of dots in the URL, and the number of symbols in the URL. They can be related to not only the whole URL but specific parts of it also like the domain, the filename, or the query. We collect several such features from the URL string itself.

5.4.2 Host Based Features:

Host based features are based on the server that hosts the webpage pointed by the URL. The simplest such feature is the IP address that the URL resolves to.

5.4.3 Domain WHOIS Based Features:

Any Regional Internet Registry (RIR) like ARIN or APNIC maintains its domain WHOIS database that contains information on the domain registrant. In addition to general information like the registrant name, the database also includes the creation and expiration dates of the domain registration. These data can be queried using the RDAP [91] protocol. We analyze these data to parse a great number of features. Examples are 'number of days before the domain registration expires' and 'number of days passed after creation of the domain'.

5.4.4 GeoIP Based Features:

GeoIP features are obtained from the IP address of the host. We used the GeoLite2 [92] IP geolocation databases from MAXMIND to collect various GeoIP features of the host IP address. GeoIP features include autonomous system number (ASN), country, city, latitude, longitude, etc.

5.4.5 N-gram Features

An n-gram is a contiguous sequence of n characters from the URL. Phishing URLs often contain common brand names (like Microsoft, Paypal) to confuse the visitor. We catch those names using the URLs n-gram.

5.5 Online Classification

Phishing URL detection is a dynamic problem. This means we need to update our classifier model periodically. However, since the URL space grows without bound we cannot train a batch learning classifier indefinitely. In contrast, an online/incremental learning [93] classifier can be updated iteratively.

If, $\mathbf{wt_{n-1}}$ is a prediction model trained with $n-1$ instances of data, an online learning algorithm can update it for the n^{th} data instance with features $\mathbf{xt_n}$ and label yt_n as follows:

$\mathbf{wt_n} = \mathbf{wt_{n-1}} + f_o(\mathbf{xt_n},\ yt_n)$, where f_o is some update function. On the other hand, a batch learning algorithm achieves the same as follows:

$\mathbf{wt_n} = f_b(\{\mathbf{xt_1},\ \mathbf{xt_2},\ \ldots\ \mathbf{xt_n}\}, \{yt_1,\ yt_2,\ \ldots\ yt_n\})$, which is computationally infeasible because n grows without bound.

For this work, we have chosen the Second Order Perceptron (SOP) online classifier from the package LIBSOL [94]. We have chosen an online classifier because we collect

new training URLs every day. It is impossible to retrain our entire model with the total dataset because the dataset grows in an unbounded manner. The training model of the online classifier can be updated with only the new URLs without retraining it from the beginning. We have also compared the accuracy of SOP with batch random forest (RF) classifier from the package scikit-learn [95].

5.6 Result

Fig. 5.2 shows the change in the accuracy and the 'receiver operating characteristic - area under the curve' (ROC AUC) with a growing sample size, for our system. As seen in the figure, for a sample size of 57 thousand, batch RF achieves an accuracy of 0.91 or 91% with a ROC AUC of 90%. Online SOP, on the other hand, achieves an accuracy of 86% for a sample size of about 96 thousand. However, we still choose the online SOP because batch RF cannot deal with the unbounded growth of the URL dataset.

It can also be interpreted from the figure that the accuracy of our classifier will increase in the future, as we get new data because the curve has not become parallel to the x-axis yet.

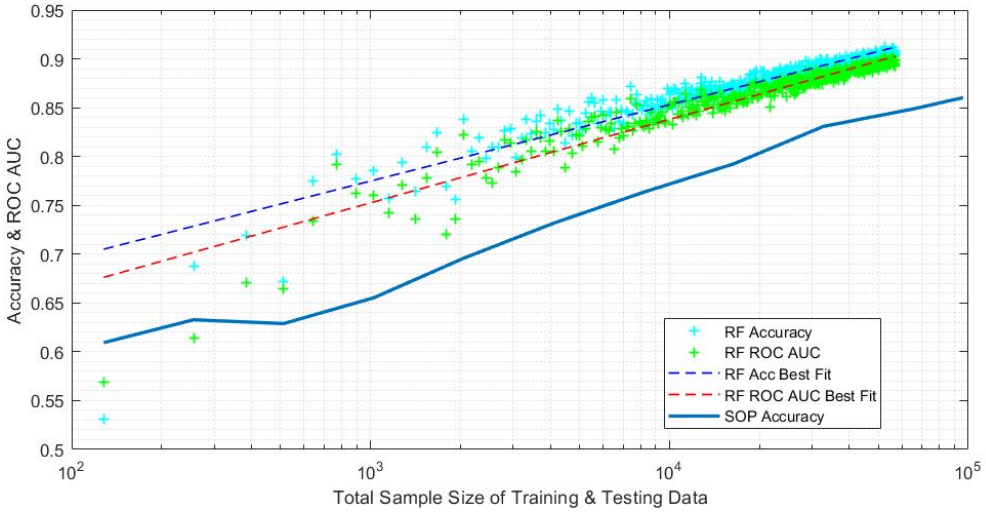


FIGURE 5.2: Accuracy & ROC AUC vs Sample Size

5.6.1 Feature Importance Analysis

We started experimentation with 142 features in total (except for N-gram features). Then we used the 'drop column' method to calculate the importance of each feature. In this method, we calculate the importance of a particular feature, by noting the change in accuracy score after removing that feature from the dataset.

The 20 most important features in our dataset are shown in figure 5.3. It is interesting to note that entropy of the URL string is the most important feature with a contribution of 2.75% in figure 5.3. This is because malicious URLs often have random characters instead of dictionary words. As a result, the entropy of those strings is higher.

The second most important feature, that we found, is the number of days since the domain registration has been updated. As malicious websites are often newer than

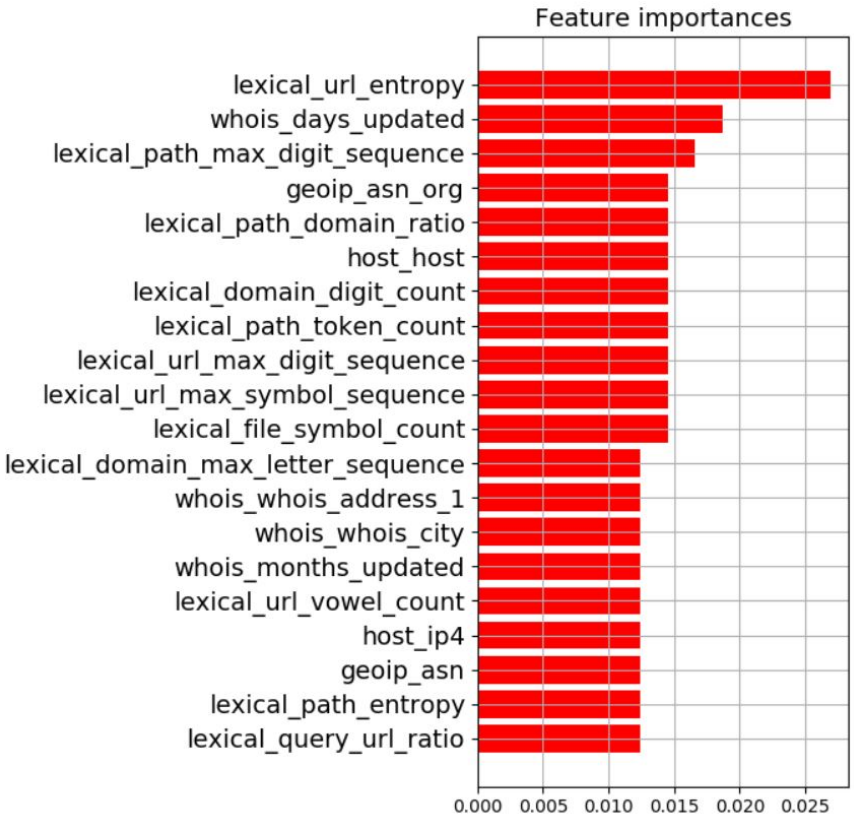


FIGURE 5.3: Top 20 Important Features

benign websites, the recently updated domains tend to be malicious. The third most important feature is the maximum number of consecutive digits in the URL path. The argument behind this is the same as the one for URL entropy. A random URL string usually has more digits than a regular string made up of words.

A key takeaway from figure 5.3 is that half of the top 20 features are lexical features. Lexical features are much less expensive to collect, making our delayed feature collection algorithm successful.

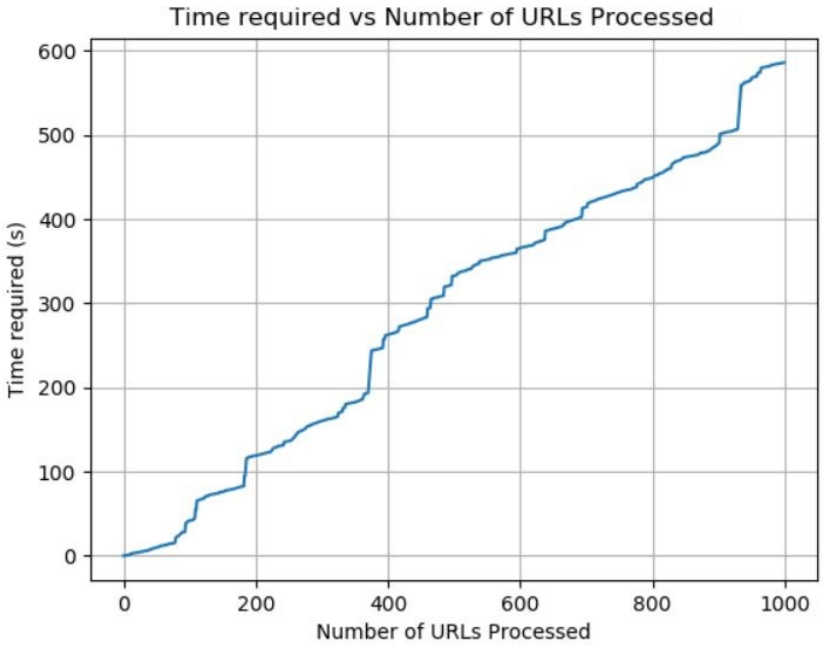


FIGURE 5.4: Linear Complexity of System

5.6.2 Cross Validation

To cross-validate our findings we have used another dataset of phishing URLs downloaded from OpenPhish.com [71]. We removed our training URLs from that dataset and ended up with 2664 URLs previously not seen by our classifier. Our classifier achieved an accuracy of 88.6% validating that our training set was not biased.

5.6.3 Complexity and Scalability

Figure 5.4 shows that the complexity of our URL classification system is linear. The jumps in the graph are attributed to offline URLs. Some of the feature collection

modules wait for timeout when an URL is offline. We can see two such timeouts at around 200s and 500s. Despite the noises, the graph is linear on average.

5.7 Conclusion

In this chapter, we have outlined a robust framework for the automated detection of phishing URLs. We have used online learning to deal with the unbounded growth of URL space. We have also incorporated selective sampling and delayed feature collection to significantly improve the system performance. Our system can detect previously unseen URLs with 86% accuracy. Moreover, we have used CYBEX-P infrastructure as a service (IaaS) to develop and deploy this system. This proves the viability of CYBEX-P for advanced threat data analysis.

Chapter 6

CYBEX-P for Modelling Attacker Behavior in Botnet

Now that we have presented an example of using CYBEX-P for advanced threat data analysis, in this chapter, we move on to a more robust approach to analyzing threat data. A major motivation behind CYBEX-P is merging heterogeneous threat data and analyzing them together. This chapter realizes that goal by developing a framework for modeling attacker behavior in a botnet.

In this discussion, we dive deep into the framework discussing the dataset which contains four different types of events including file downloads, SSH logins, network traffic, and command input into the shell. We then discuss why and how the dataset is parsed into TAHOE and the advantages of using TAHOE. Finally, we validate the

framework with extensive experimentation. This framework is a novel contribution that proves the feasibility of CYBEX-P for robust threat data analysis.

6.1 Introduction

A key component in many types of cyberattacks is a bot [96] – a malicious program that allows an attacker to remotely control the infected host. Some notable examples of bot malware are Mirai [97], Torpig [98], Conficker [99] etc. A bot also connects the infected host to a botnet [100, 101] - a network of such hosts. Botnets make up a large portion of the cybersecurity market.

Attackers use various techniques to infect a host with bot malware. For example, they can send the malware as an email attachment, post the download link on online forums and social networks, or host it on a website for drive-by downloads. They can also directly perform a brute-force attack to crack the password of a host.

Irrespective of how the host is infected, the attackers usually gain access to the 'shell' of the compromised hosts. Therefore, they can use the botnet for a variety of cyberattacks such as adware distribution, DDoS attacks, hosting phishing websites, ransomware distribution, sending spam emails, spamming search engines, stealing credit card info, etc. Thus, it is desirable for any person or any organization to detect and block a botnet in their network.

The increasing adoption of the Internet of Things (IoT) has made IoT devices a major target of bots [102]. The most prominent example is the Mirai botnet [97] which compromises IoT devices using brute-force attacks on the login credentials. It was first discovered in late 2016 and is still the most widespread botnet plaguing the IoT network.

IoT botnets, like Mirai, were able to take the internet by storm because of the proliferation of weakly configured IoT devices. A large number of IoT devices like refrigerators and CCTV cameras are configured with easily guessable usernames and passwords. Bot malware exploits that to perform a dictionary attack on the username/password pairs to gain access to the shell.

The simplest defense against IoT botnets is manually blacklisting the IP addresses of the infected hosts. However, numerous hosts are compromised every day. At the same time, many compromised hosts become benign every day, as the owner regains control. Consequently, it is impossible to list all of their IPs. Moreover, blacklisting is a reactive approach because honeynets can. An IP shows up on a blacklist only after the host has done some harm. As a result, the industry would greatly benefit from a proactive defense mechanism against botnets. An intelligent system should detect a zero-day bot-host from its behavior, not the IP. If a bot-host is detected in an early phase of the kill chain, it cannot do any harm to anybody.

Meanwhile, Intrusion detection systems (IDS) [103] use network signatures to detect

bots. While they work well for known patterns they cannot adapt to the new attacks. It also takes a long time to detect an attack pattern, analyze it and create its signature before adding it to an IDS [104]. Another popular approach is using anomalies in network traffic [105–107] to detect bots. Some works take it further to detect anomalies in DNS traffic [108–111].

The final approach is detecting anomalies in the infected hosts [112–114]. The parameters used by these works are system calls, system API calls, syslogs, event logs, etc. However, to the best of our knowledge, no previous work has considered heterogeneous threat data to do behavior modeling. In particular, no previous work has considered correlating network traffic data, file download data and commands input into the shell to model attacker behavior. In addition, we have identified the following challenges in modeling attacker behavior in a botnet:

1. The phrase 'attacker behavior' is not well defined.

2. There is no standard process or structure for modeling attacker behavior in a botnet.

3. There is little automation in attacker behavior analysis, from data collection to data analysis.

4. No previous work has considered multiple heterogeneous sources of threat data in modeling attacker behavior.

5. Limited number of works have considered commands input into a compromised shell for modeling attackers.

This work considers heterogeneous threats for attacker behavior modeling, including network traffic, commands input into a compromised shell, and files downloaded into the host. To the best of our knowledge, no previous work has considered heterogeneous data for attacker behavior modeling. This is the novel contribution of this work. The contributions of this work are summarized below:

1. We have collected a large dataset of heterogeneous threat data from bot-infected hosts.

2. We have clearly defined 'attacker behavior' in this paper as a 4 element vector.

3. We have automated the whole process from data collection to analysis using CYbersecurity information Exchange (CYBEX) [44, 115].

4. We have integrated multiple sources of threat data including network traffic, file downloads, and shell commands.

5. We have shown the efficacy of Temporal Convolutional Network (TCN) [116] in predicting attacker behavior and compared it with Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU).

6. We have demonstrated the validity of our data model and TCN by predicting attacker behavior with an accuracy between $85 - 97\%$.

6.2 Background

6.2.1 Honeypot

A honeypot is a decoy system. The Honeynet project [117] defines a honeypot as *A security resource whose value lies in being probed, attacked, or compromised.* As honeypots have no production value, any activity logged in a honeypot can be deemed malicious.

Honeypots can be classified into two categories based on their purpose:

1. Production honeypot: Production honeypots are used in campus networks to lure attackers away from production machines. They can also be used to detect attacker IP addresses or email addresses. They protect production servers by posing themselves as easy targets for attackers.

2. Research honeypot: Research honeypots are used to collect information. This information is further analyzed to detect new tools and techniques, to understand the behavior of attackers, and to detect attack patterns. Finally, the analyzed data can lead to newer defense techniques.

Honeypots can be further classified into three categories on their level of interaction with the attacker:

1. High Interaction Honeypots: Simulates all aspects of a real operating system (OS) completely. These honeypots can collect more information. However, they are riskier to maintain, because the attacker can launch further attacks from these.

2. Medium Interaction Honeypots: Simulates the aspects of an OS that cannot be used to launch further attacks.

3. Low Interaction Honeypots: Simulates very basic aspects of an OS. They can collect very limited information and are low risk.

6.2.2 Cowrie Honeypot

Cowrie [118] is a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system. In this work, we use Cowrie in the medium interaction mode.

In our setup, cowrie only allows SSH logins into our honeypot. An attacker can log in to the system using any username and password combination. Cowrie logs all the interactions including the source IP address of the attacker, the SSH parameters, and the commands input while the attacker is logged in.

6.3 Dataset

6.3.1 Data Source – Cowrie Honeypot

Cowrie [118] is a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system. In this work, we use Cowrie in the medium interaction mode.

We have set up 5 instances of Cowrie around the world – Amsterdam, Bangalore, London, Singapore, Toronto. The honeypots are configured to allow only SSH logins into the system. An attacker can log in to the system using any username and password. Cowrie logs all the interactions including the source IP of the attacker, the login credentials, the SSH parameters, the downloaded files, and the commands input into the shell.

6.3.2 Cowrie Data as Events

```
{
  "eventid": "cowrie.session.file_download",
  "timestamp": "2020-04-28T00:00:22.134604Z",
  "src_ip": "5.188.87.49",
  "session": "d151a9c7",
  "sensor": "london",
  ...
}
```

FIGURE 6.1: Common attributes of all Cowrie events.

Cowrie structures collected data into events. Fig. 6.1 shows the common attributes of a Cowrie event. These attributes are explained below –

1. *eventid:* Denotes the type of the event. The 'eventid' *cowrie.session.file_download* in Fig. 6.1 means, it was generated when the attacker downloaded a file into the compromised machine.

2. *timestamp:* The time when the event was recorded by the honeypot.

3. *src_ip:* Source IP address of the attacker.

4. *session:* A session is a sequence of events generated during one login session. Cowrie maintains a unique 'session' for each session and assigns the same session-ID to the events of a session.

5. *sensor:* Hostname of the honeypot server.

6.3.3 Cowrie Event Types

Cowrie generates about 20 different 'eventid's. Many of these event types are related to the SSH session, key exchange, and logging and do not carry valuable information. For this work, we are interested in the following 'eventid's –

- *cowrie.login*: Generated when the attacker tries to SSH into the host. Contains the username and the password of the SSH request. Fig. 6.2 shows a *cowrie.login*

event. Here the suffix *.success* means that the login attempt was successful. Note that the common attributes shown in Fig. 6.1 are omitted from Fig. 6.2.


```
{
  "eventid": "cowrie.login.success",
  "password": "asdasdasd",
  "username": "root",
  ...
}
```


FIGURE 6.2: Attributes of `cowrie.login` event.

- *cowrie.direct-tcpip*: Generated when the attacker tries to communicate over the internet through the TCP-IP protocol. Contains the destination IP, destination port and the data (if present). The suffic *.data* means that this communication contains data. Fig. 6.3 shows a *cowrie.direct.tcp-ip* event. Note that the 'data' field is truncated. Also, the common attributes from Fig. 6.1 are omitted.


```
{
  "data": "\\x03\\x00\\xa6...",
  "dst_ip": "www.walmart.com",
  "dst_port": 443,
  "eventid": "cowrie.direct-tcpip.data",
  ...
}
```


FIGURE 6.3: Attributes of `cowrie.direct.tcp-ip` event.

- *cowrie.session.file_download*: Generated when the attacker downloads a file into the compromised host. Contains the download URL, the file hash and the actual binary of the file. Fig. 6.4 shows a *cowrie.session.file_download* event.

- *cowrie.command*: Generated when the attacker inputs a command into the shell of the compromised host. It contains the exact command. The suffix *.success* means the command was simulated successfully. Fig. 6.5 shows a

```
{
  "eventid": "cowrie.session.file_download",
  "outfile": "dl/6e223babfbd3e...",
  "shasum": "6e223babfbd3eef8...",
  "url": "http://192.210.236.38/bins.sh"
  ...
}
```

FIGURE 6.4: Attributes of `cowrie.session.file_download` event.

cowrie.command event. The *input* attribute contains the exact command input by the attacker in the shell.

```
{
  "eventid": "cowrie.command.success",
  "input": "cat /proc/cpuinfo",
  ...
}
```

FIGURE 6.5: Attributes of `cowrie.command` event.

6.3.4 Command, Parameter & Type

After logging in, the attackers often execute different commands in the honeypot shell. These commands are documented by Cowrie in the *cowrie.command* events. An example command is `wget NasaPaul.com/v.py`. Here `wget` is the actual command and `NasaPaul.com/v.py` is its parameter.

In our database, we have seen about 40 unique commands. However, many of these commands, like `wget` take parameters and there are more than 300 unique command-parameter combinations. We have further classified these commands into 7 types based on the intention of the attacker. These types are:

1. System Info – Check software, hardware, or system configuration.

2. Cover Track – Hide evidence of intrusion and malicious activity.

3. Install – Install a software in the system that was not previously in there.

4. Download – Download a remote file into the honeypot system.

5. Run – Run or execute a program or a script.

6. Escalate privilege – Change password or gain root access to the system.

7. Change config – Change system configuration including hostname, network, and firewall configuration.

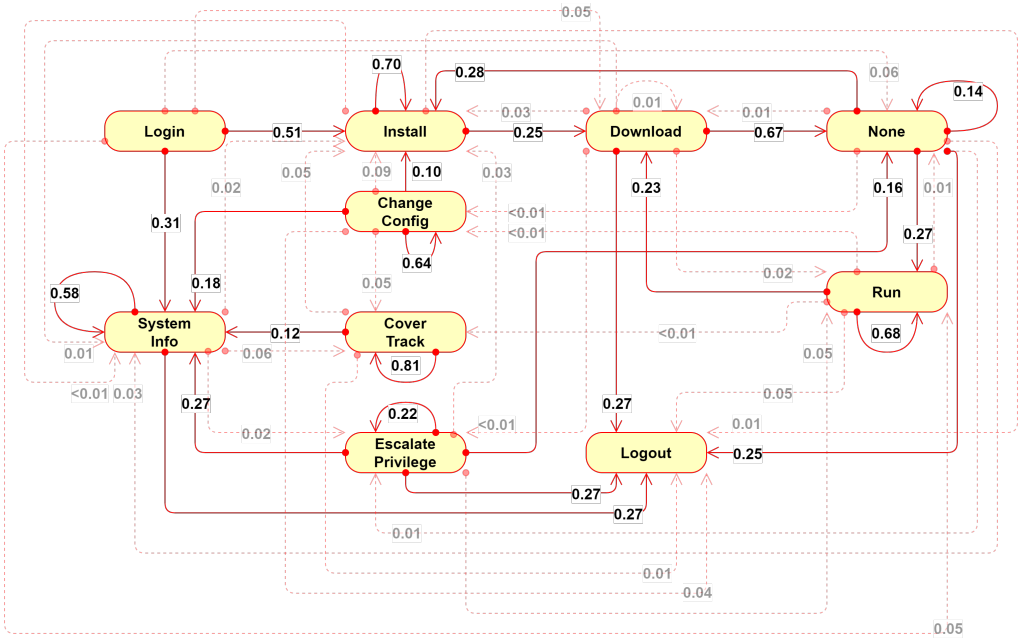


FIGURE 6.6: Representation of Cowrie sessions as a finite state machine of the command types.

Fig. 6.6 shows the Cowrie sessions as a state machine of these command types. The edges are the probabilities of transition from one state to the next. The state `None`

means the event is not of the type shell command. Table 6.1 shows the different command types and corresponding commands. The classification of commands into command types is further visualized in Fig. 6.7.

TABLE 6.1: Classification of commands into different types.

Command Type	**Commands**
System Info	cat, echo, free, help, history, last, ls, ps, w, grep, lscpu, nproc, uname, wl
Cover track	export, reboot, rm, touch, unset
Install	apt, apt-get, install, yum
Download	scp, wget
Run	nohup, perl, python, /tmp/*, /usr/*
Escalate privilege	ln, mkdir, mv, passwd, su, sudo
Change config	hostname, ifconfig, /ip, kill, susefirewall2, service

6.3.5 Dataset at a Glance

1. Total number of events = 3567
2. Total number of sessions (sequences) = 393
3. From date = April 4, 2020
4. To date = May 8, 2020
5. Number of Cowrie honeypots = 5
6. Location of honeypots = Amsterdam, Bangalore, London, Singapore, Toronto

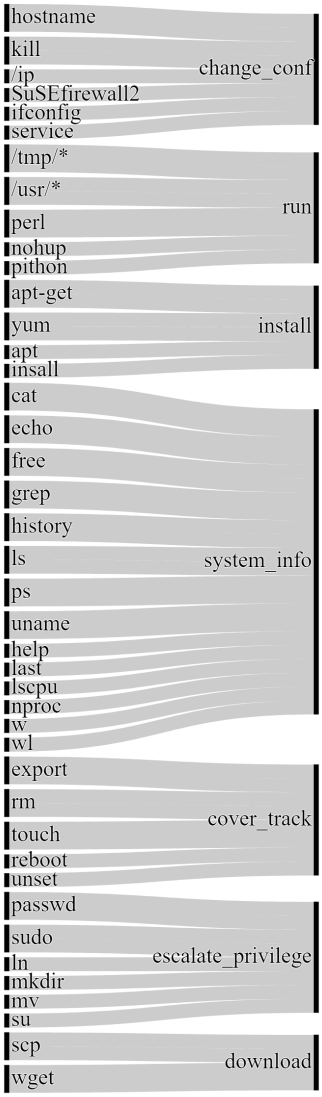


FIGURE 6.7: Map of commands to command types.

6.4 Data Processing

As discussed in section 6.3, cowrie generates heterogeneous data with different attributes. For example, a *cowrie.direct-tcpip* event has the *dst_ip* and *dst_port* attributes, which other event types do not have. In this section, we extract features

from such heterogeneous and group them into a single feature table so that they can be fed into a learning model.

Along with CYBEX, we have further developed TAHOE a graph-based cyberthreat language (CTL). In this section, we discuss the modeling of Cowrie data in TAHOE format. TAHOE offers several advantages over traditional CTLs – Firstly, TAHOE can store all types of structured data. Secondly, queries in TAHOE format are faster than in other CTLs. Thirdly, TAHOE intrinsically correlates the heterogeneous data with itself. Finally, TAHOE is scalable for all kinds of data analysis - a major limitation of other CTLs.

In this section, we further discuss how to featurize such heterogeneous data to use them for machine learning. We start by explaining the complete lifecycle of the data in CYBEX from data generation.

6.4.1 Data Flow in CYBEX

6.4.1.1 Data Generation

Each Cowrie honeypot (2 in Fig. 3.2) simulates a generic IoT device. They generate data in the format of Fig. 6.1. The honeypots log these data pieces into a file in the respective server. We call each such log message a raw document.

6.4.1.2 Data Input

Each of our honeypot installations have a connector agent (❷ in Fig. 3.2). The connector is a script that reads the raw data from log files and sends them to the CYBEX collector (❸) via a real-time websocket. The data in transport are encrypted via TLS.

6.4.1.3 Data Collection

The collector then posts the data to the API (❹). The API encrypts the data with the public key of the archive cluster (❻.❶) and stores the encrypted data in the cache data lake (❺).

6.4.1.4 Data Archiving

```
{
  "itype": "raw",
  "data": {
    "eventid": "cowrie.session.file_download",
    "timestamp": "2020-04-28T00:00:22.134604Z",
    "src_ip": "5.188.87.49",
    "session": "d151a9c7",
    "sensor": "london",
    ...
  },
  "sub_type": "cowrie_honeypot",
  "timezone": "US/Pacific",
  "_hash": "3d5792b...",
  ...
}
```

FIGURE 6.8: A Cowrie event encapsulated in a TAHOE `raw` document.

The archive cluster (6.1), then pulls the data from the cache data lake (5), decrypts the data using its private key, converts the cowrie events into TAHOE `raw` format and stores them in the archive database (7). TAHOE `raw` puts a wrapper around the Cowrie event.

Fig. 6.8 shows a TAHOE `raw` document. The `_hash` (truncated in figure) is the unique ID of the document and generated as the SHA256 checksum of the `data` field. CYBEX collects different types of data; so TAHOE uses the `sub_type` field to distinguish between them.

6.4.1.5 Data Filtering

Data filtering in CYBEX means parsing a TAHOE `raw` document into TAHOE `events`. Each `sub_type` of a TAHOE `raw` document represents a different type of data; and has its own *filter* scripts. The *filter* basically extracts different attributes from the `raw` document and restructures them into TAHOE `events`.

This parsing is done by the analytics cluster (6.3). It reads the `raw` data from the archive database (7), parses the data, and writes the results back in the archive database. Fig. 6.9 shows the structure of a TAHOE `event`.

The `sub_type` of a TAHOE event depends on the *eventid* of the original `raw` document. For example a *cowrie.login* event is parsed into a TAHOE `ssh` event. The mapping is – *cowrie.command* ⇒ `shell_command`, *cowrie.direct.tcp-ip* ⇒ `network_traffic`,

```
{
  "itype" : "event",
  "timestamp" : 1588093536.969,
  "category" : "unknown",
  "data" : {
    "success" : [false],
    "shell_command" : ["cat /proc/cpuinfo"],
    "attacker" : [{
        "ipv4" : ["134.122.20.113"]
    }]
  },
  "_cref" : [
    "e7dc7351c504da69f7a43421...,
    "966fca3ed576e47e9d2ae2a7...,
    "a58a2e656c004f01b38dc77c...
  ],
  "sub_type" : "shell_command",
  "_hash" : "b3da61a6313307f739...",
  ...
}
```


FIGURE 6.9: A TAHOE `event` document.

cowrie.session.file-download $\Rightarrow$ `file_download`, *cowrie.login* $\Rightarrow$ `ssh`. So, TAHOE basically normalizes the Cowrie events into the standardized format - TAHOE.

Notice that, the Cowrie *session* ID is not stored in TAHOE `events`. For that, we use a separate TAHOE structure called a `session`. Fig. 6.10 shows the structure of a TAHOE `session`. The field `_ref` stores the ID of all the `events` that belong to this session. So, basically it forms a directed graph with the `session` node as the root and the `events` as leaves.

This concludes the default flow of any threat data through CYBEX. We have now converted Cowrie events into TAHOE events without any loss of information. Next, we begin processing the data for this particular task of attacker behavior modeling.

```
{
  "itype" : "session",
  "data" : {
    "hostname" : ["london"],
    "sessionid" : ["5a0facf9"]
  },
  "_cref" : [
    "53df245bcefb3f2a558349c37...",
    "39885eec34b95fa2acdfffd14..."
  ],                                ..."
  "_ref" : [                        ..."
    "b3da61a6313307f7394510146...",
    "1e31784145b52a42d964f5a5c...",
    "cb3be781b29297571cc20cbf8...",
      ...                           ..."
  ],                                ..."
  "sub_type" : "cowrie_session",..."
  "_hash" : "98601c106789882a4ee...",
  "start_time" : 1588924147.615,
  "duration" : 3.29502511024475,
  "end_time" : 1588924147.616
}
```


FIGURE 6.10: A TAHOE `session` document.

6.4.2 Advantages of using CYBEX & TAHOE

The data collected from Cowrie honeypot is used as an example to validate our methodology in this work. However, we want to propose this methodology for all types of threat data collected from heterogeneous sources. This is particularly problematic because different sources store or log data in their format. For example, two firewalls from two different vendors will collect network traffic logs in different formats. CYBEX automatically recognizes the sources and normalizes those seemingly different data into TAHOE. TAHOE acts as the standardized format here while CYBEX acts as the automated parser.

Each event has a different set of attributes or properties. This makes them unsuitable for storing as a row in a relational database. TAHOE uses a JSON structure to store

such arbitrarily structured events. Moreover, there could be any number of edges or connections between the attributes, objects, events, and sessions. Such arbitrary length of the edges array is again unsuitable to be stored in a cell in a relational database. However, the JSON structure poses no such limitation on TAHOE. Finally, TAHOE differs from other JSON based CTI (e.g. STIX) by being indexable. As a result, we can query events connected to an attribute or a group of events connected to a session fast. To the best of our knowledge, there is no such CTI available in the industry right now.

6.4.3 Sequence of Events

In subsection 6.4.1 we saw how CYBEX parses any threat data into TAHOE events. Now, we further curate these TAHOE events for the task at hand - attacker behavior modeling. In this work, we do not consider each event independent. Rather we are interested in modeling attacker behavior as a sequence of events.

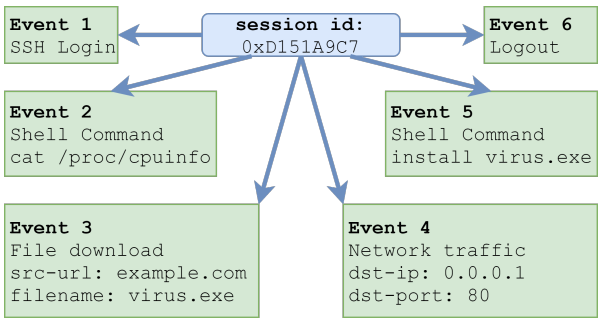


FIGURE 6.11: A TAHOE `session` with 6 `events` as a directed graph.

As stated in subsection 6.3.2, Cowrie generates a unique *session* ID whenever an attacker logs into the honeypot. Cowrie stores this ID in all events generated during

this login session. We can use that ID to group the events in a session. We can then sort these events by their timestamps to form a sequence.

Fig. 6.11 shows an example sequence of events. Here the attacker logs into the honeypot, executes a command in the shell, downloads a file, sends some data over TCP/IP, executes another command, and then logs out. There are 6 events in this example session. However, there can be any number of events in a session. In our dataset, we have seen a minimum of 2 events and a maximum of 47 events in some sessions.

At this point, we group all events into event sequences like that of Fig. 6.11. We then end up with several sequences, each with an arbitrary number of events. We then move on to extracting features for each of these events.

6.4.4 Feature Extraction

Now, for each event in a session we extract the following features:

Time related

1. Hour (integer): Hour of day.
2. Date (integer): Date of month.
3. Month (categorical): January, February etc.

4. Day (categorical): Sunday, Monday etc.

Session Related

5. Session start (boolean): Is this event at the beginning of a session?

6. Session end (boolean): Is this event at the end of a session?

7. Event order (integer): How many events have been recorded in this session?

Other common features

8. Event type (categorical): Valid event types are ssh, network traffic, shell command, and file download as described in subsubsection 6.4.1.5.

9. Sensor (categorical): Location of the honeypot server - Amsterdam, Bangalore, London, Singapore, or Toronto.

10. Attacker IP (categorical): IP address of the attacker.

Only for `cowrie.direct.tcp-ip` events

11. Destination IP (categorical): Destination IP address of the TCP-IP packet/s.

12. Destination Port (integer): Destination port of the TCP-IP packet/s.

Only for `cowrie.command` events

13. Command + parameter (categorical): The shell command with parameter.

14. Command (categorical): This feature is derived from 'command + parameter' and lists the actual shell command without parameter.

15. Command type (categorical): This is another derived feature and lists the type of command as described in subsection 6.3.4.

16. Command success (boolean): Was the command successfully simulated?

Only for `cowrie.login` events :

17. Login success (boolean): Was the attacker successful to log in?

For example, if we extract the features of the event in Fig. 6.9, we get the feature vector shown in Fig. 6.12. Note that, it does not have any valid value for the features 'Dest IP' and 'Dest Port', because these two features are defined for 'network traffic' events only. Similarly, 'login success' is defined for 'ssh' events only. Also note that the 'session start', 'session end', 'event order', and 'sensor' features are not directly extracted from the event data in Fig. 6.9. Rather these are extracted from the session data in Fig. 6.10. We can look this up in our database, because the session in Fig. 6.10 contains the `_hash` of this event in its `_ref` field.

```
Common features:
================

Hour  Date  Month  Day  Session   Session  Event    Event  Sensor        Attacker
                        Start       End  Order     Type                       IP
----  ----  -----  ---  -------  --------  -----  -------  ------  --------------
  17    28    Apr  Tue     True     False      1    Shell  London  134.122.20.113
                                                   Command


Features related to a particular event type:
============================================

Dest  Dest          Command +  Command  Command  Command    Login
  IP  Port          Parameter           Type  Success  Success
----  ----  -----------------  -------  -------  -------  -------
None  None  cat /proc/cpuinfo      cat   System    False     None
                                          Info
```

FIGURE 6.12: Feature vector of the event in Fig. 6.9.

At this point, we have several sequences like that of Fig. 6.11. Each sequence has an arbitrary number of events and each of those events is represented by a 17 parameter vector-like Fig. 6.12. With this dataset, we are ready to define the problem statement of our analysis methodology.

6.5 Analysis Methodology

So far we have modeled attacker behavior as a sequence of events. We have also represented each of those events as a 17 element vector. In this section, we assess the validity of our model with real data. We do this by predicting future attacker behavior based on past events. If we can successfully predict the next step an attacker takes, we can simultaneously infer that attacker behavior is predictable and our model is valid.

To show this, we have chosen to predict the following targets – (1) event type (4 different values), (2) shell command with parameter (300+ different values), (3) shell command (40+ different values) and (4) shell command type (7 different values). We call this set of targets the 'attacker behavior' for a particular event.

This is a sequence modeling problem because an event in a sequence depends on the previous events. Recurrent neural networks (RNN), like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), are better suited for sequence modeling [119]. However, a recent publication [116] has shown the viability of a convolutional neural network called a temporal convolutional network (TCN) as well. So, in this section, we compare three neural networks to predict attacker behavior – (1) TCN, (2) LSTM, and (3) GRU.

6.5.1 Problem Statement

Here, the set of predictors is $\mathbb{P}$ = {hour, date, month, day, session start, session end, event order, event type, sensor, attacker IP, dest IP, dest port, command + parameter, command, command type, command success, login success}. And, the set of targets is $\mathbb{T}$ = {event type, command + parameter, command, command type}

Now, let us assume, $\mathbb{S} = S_1, S_2, ..., S_N$ is a set of N sequences. Here, $S_i =< E_1, E_2, ..., E_{M_i} >$ is the i^{th} sequence with M_i events in it.

Then, $\mathbb{P}_{i,j} =< x1_{i,j}, x2_{i,j}, ..., x17_{i,j} >$ is the vector of the 17 features in $\mathbb{P}$ for the j^{th} event in the i^{th} sequence. This vector contains our predictors.

Also, the target vector $\mathbb{T}_{i,j} =< z1_{i,j}, z2_{i,j}, ..., z4_{i,j} >$ is the vector of the 4 targets in $\mathbb{T}$ for the j^{th} event in the i^{th} sequence. $\mathbb{T}_{i,j}$ represents attacker behavior in our work.

We want to find the function f which minimizes some expected loss between the targets and the predicted values $\mathcal{L}(\mathbb{T}, f(\mathbb{P}))$.

6.5.2 Temporal Convolutional Network (TCN)

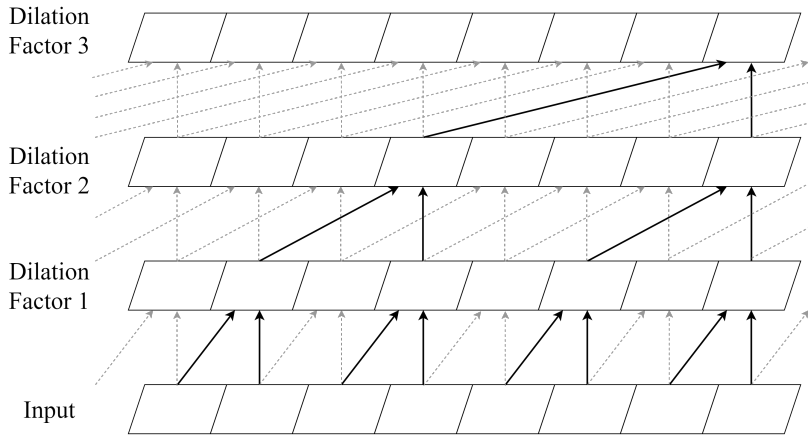


FIGURE 6.13: Dilated causal convolutional layers of a typical TCN.

Convolutional neural networks (CNNs) are commonly associated with image classification tasks. However, Bai et al. [116] outlined the general structure for a temporal convolutional network (TCN) which can be used to create a robust prediction model for sequences. They have also empirically shown how TCN matches or even outperforms traditional recurrent neural networks (RNNs) in sequence modeling and prediction.

The principal building block of TCN is a dilated causal convolution layer. Here, 'causal' means the output for the current step does not depend on future steps. Also, dilated convolutions are used to increase the receptive field of the layers. Multiple such layers can be stacked to form a deeper network. The dilation factor is increased exponentially as shown in Fig. 6.13.

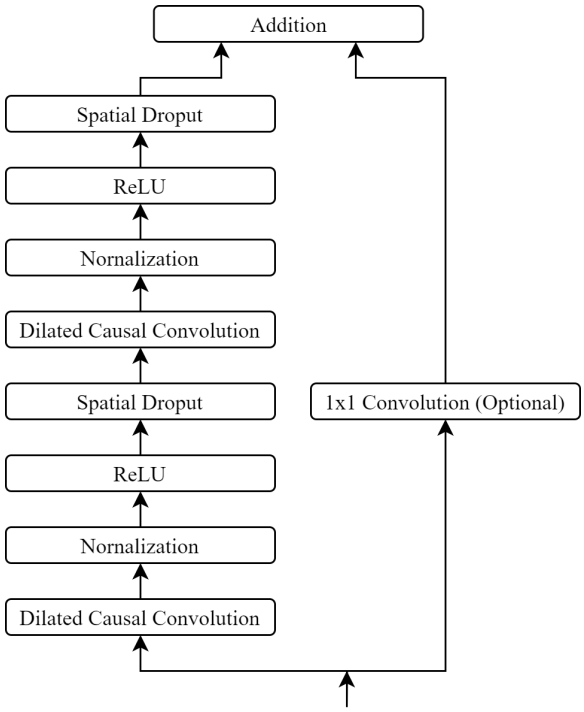


FIGURE 6.14: A TCN Residual Block.

The architecture of a general TCN described in [116] contains multiple residual blocks. Each residual block consists of two dilation causal convolution layers with the same dilation factor along with normalization, ReLU activation, and dropout layers. The input to each residual block is also added to the output when the number of channels between the input and the output is different. A general residual block is shown in Fig. 6.14.

FIGURE 6.15: General architecture of a TCN Classifier.

We can then put one or more such residual blocks into a general sequence classifier to get the architecture of a TCN classifier as shown in Fig. 6.15. The network begins with a sequence input layer followed by one or more residual blocks. The residual blocks are then followed by a fully connected layer, a softmax layer, and a classification output layer.

6.5.3 Long Short-Term Memory (LSTM)

LSTM [120] is an artificial recurrent neural network (RNN) architecture [121] used by deep learning practitioners for sequence modeling. LSTM has feedback connections in contrast to traditional feed-forward neural networks. As a result, LSTMs can learn long-term dependencies. This property makes LSTM suitable for sequence modeling and predictions.

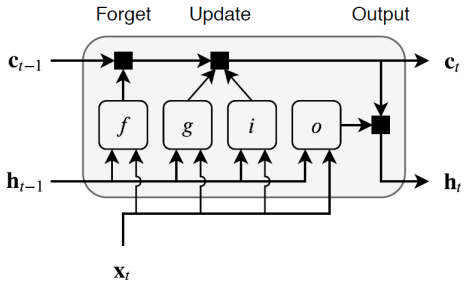


FIGURE 6.16: An LSTM Block.

The core component of an LSTM network is an LSTM block as shown in Fig. 6.16. Here, c_t is the cell state at time-step (sequence step) t whereas h_t is the hidden state also called the cell output. The forget gate, f, determines which values to remove from the cell state, whereas the input gate, i, controls which value to update. The actual update values are determined by the memory gate, g. Finally, the output gate, o controls which value to output.

Each element of a sequence passes through the LSTM block and updates it, forming an LSTM layer. Just like TCN, we can place this LSTM layer inside a general sequence classifier to get the architecture of an LSTM classifier as shown in Fig. 6.17. We have added a dropout layer after the LSTM block in Fig. 6.17 to avoid overfitting in the network.

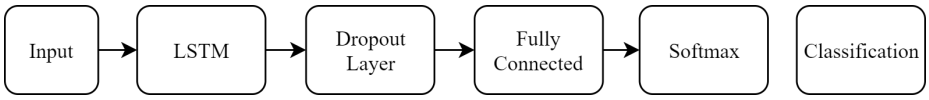


FIGURE 6.17: A general LSTM Classifier.

6.5.4 Gated Recurrent Unit (GRU)

A GRU [122] is constructed exactly like an LSTM network, except for the output gate. A comparable GRU typically has fewer trainable parameters because it lacks the output gates. It also converges faster for the same reason. GRU has comparable performance to LSTM for a majority of sequence modeling tasks and sometimes outperforms LSTM for less repetitive sequences. The general architecture of a GRU

classifier is the same as that of an LSTM classifier except it has a GRU block in place of the LSTM block.

6.6 Result & Validation

In this section, we validate our model by predicting attacker behavior. As mentioned in section 6.5, we predict the 4 features event type (4 class labels), shell command with parameter (300+ class labels), shell command (40+ class labels), and shell command type (7 class labels) for the next event in a session. We call this set of targets the 'attacker behavior'. We compare the accuracy and performance of 3 neural networks – TCN, LSTM, and GRU in this section. The design, simulation, and testing of the neural networks are done in Matlab [123].

As listed in subsection 6.3.5, for this simulation, we have collected a robust dataset of 3567 'cowrie' events. These events belong to 393 'cowrie' sessions and span over a duration of 1 month from 4 April 2020 to 8 May 2020. We have collected the dataset from 5 different 'cowrie' honeypots placed all over the world. The locations of the honeypots are – Amsterdam, Bangalore, London, Singapore, and Toronto.

To optimize the parameters of the neural networks, we have used a simple grid search. The results of the grid search for TCN are shown in subsection 6.6.1.

For the first test, we have randomly split the 393 sessions in a ratio of 80 : 20 into training and test subsets. Note that, each session is considered one sequence in our

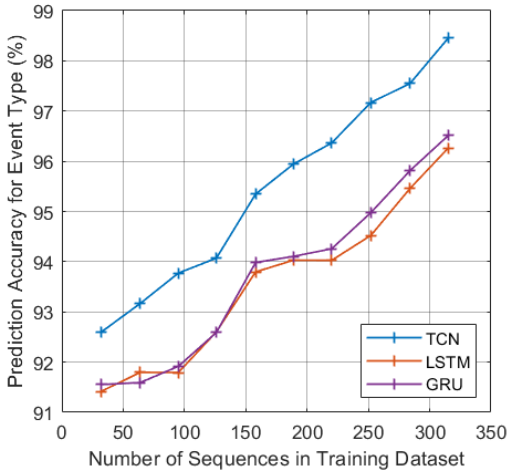
TABLE 6.2: TCN v LSTM v GRU.
training sequences = 315; # test sequences = 78.

	Accuracy (%)		
Target	TCN	LSTM	GRU
Event type	**98.45**	96.26	96.51
Command + paramter	**85.58**	80.55	81.79
Command	**94.64**	88.87	87.75
Command type	**96.37**	91.63	90.93

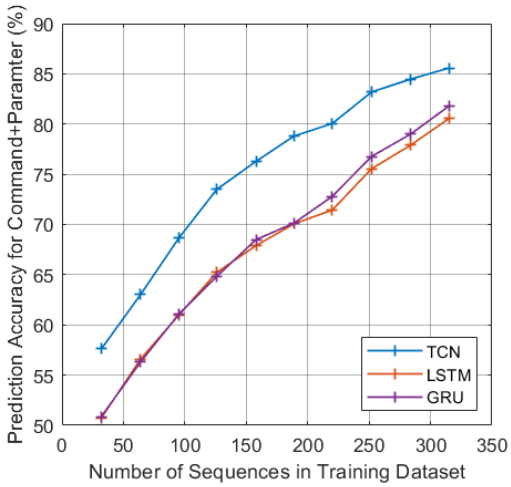
model and they have a variable number of events in them. So, we ended up with 315 sequences (2839 events) in the training dataset and 78 (728 events) sequences in the test dataset. Then, we have trained our models on the training dataset. Finally, we have tested the accuracy of TCN, LSTM, and GRU in predicting the 4 targets for the next event in the sequence. The prediction accuracy-values are listed in Table 6.2. It's seen that LSTM and GRU have comparable performance, while TCN largely outperforms the other two in all 4 cases.

While at first glance the accuracy for 'command + parameter' seems low, at 85%; it should be noted that this label assumes a different value for a different parameter supplied to the same command. For example, `mkdir temp1` and `mkdir temp2` will be registered as different classes for this target event though they are the same commands and serve the same purpose.

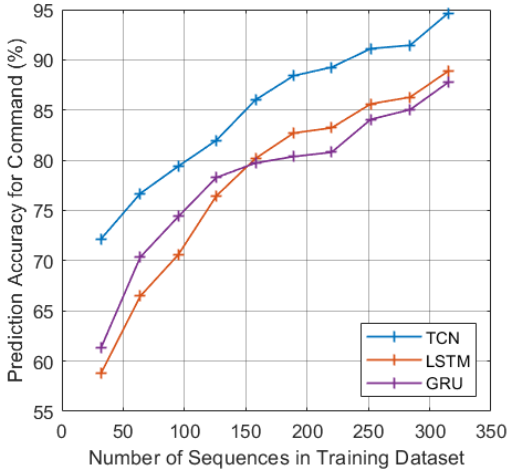
Next, we have tested the change of accuracy with the size of the training set. We have done this for all of the four targets and the results are shown in Fig. 6.18. In general, the training accuracy increases with the number of training samples in all cases for all the algorithms. And just like before, LSTM and GRU perform comparably while



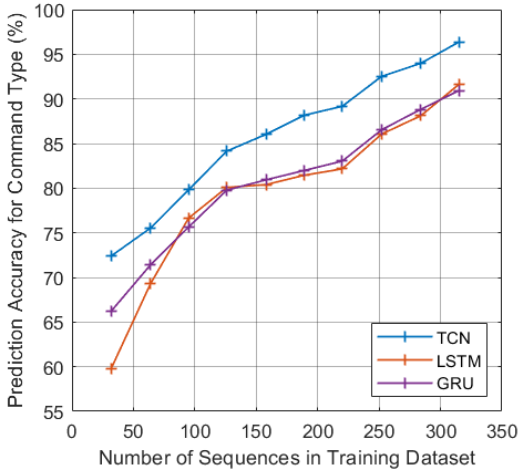
(A) Target = Event type.



(B) Target = Command + parameter.



(C) Target = Command.

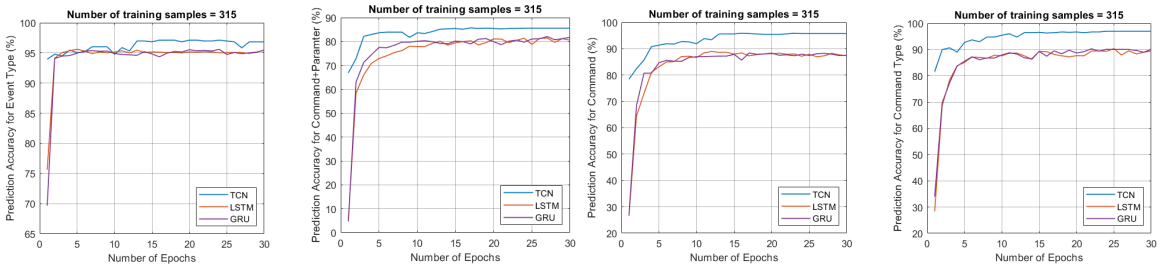


(D) Target = Command Type.

FIGURE 6.18: Accuracy vs Number of Sequences for TCN, LSTM and GRU

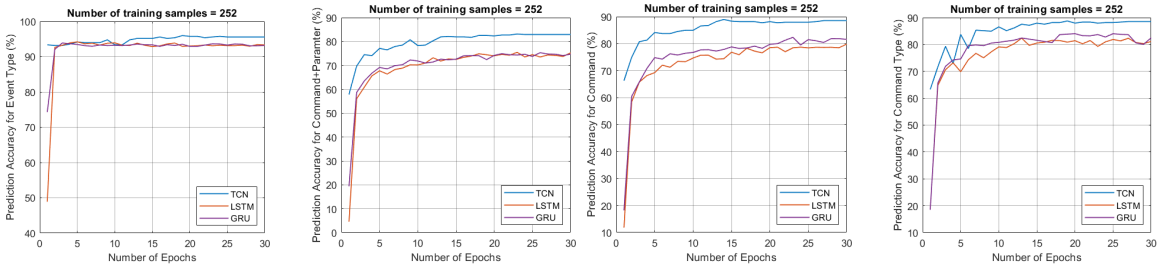
TCN outperforms both of them by a large margin.

Finally, we have tested how fast each algorithm converges to the final accuracy. We have again done this for all four targets. We have also compared the results for two different numbers of training sequences – 315 and 252. The results are shown in figures 6.19 and 6.20. These results show that TCN converges perfectly to the final accuracy well before the maximum 30 epochs in all cases. LSTM and GRU, however,



(A) Event type (B) Command+param (C) Command (D) Command Type

FIGURE 6.19: Accuracy vs Epochs for TCN, LSTM and GRU. # of training sequences = 315.



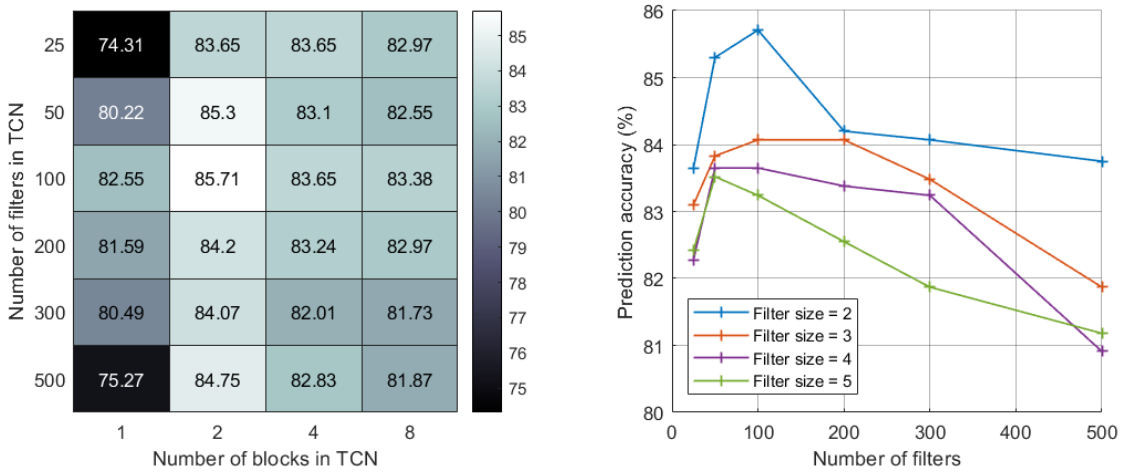
(A) Event type (B) Command+param (C) Command (D) Command Type

FIGURE 6.20: Accuracy vs Epochs for TCN, LSTM and GRU. # of training sequences = 252.

perform significantly worse than TCN.

6.6.1 Optimization of TCN Parameters

This subsection includes the results of the grid search that we used to optimize the TCN parameters. The number of filters per block, the number of residual blocks, and the filter size are the three major parameters of a TCN which in turn determine the number of learnable parameters of the network. Fig. 6.21a shows the accuracy of our network, as a heatmap, for different combinations number of blocks and filter size. As evident from Fig. 6.21a the model performs best for 2 residual blocks with 100 filters in each residual block. Furthermore, we plotted the accuracy of the network



(A) Heatmap of accuracy for different values of `numBlock` and `numFilt`.

(B) Accuracy vs number of filters for varying filter size.

for varying filter sizes in Fig. 6.21b. Fig. 6.21b shows that filter size 2 is the optimal choice for this problem irrespective of the number of filters in each residual block.

6.7 Parameters of Models

.

TCN Parameters	
Number of residual blocks	2
Number of filters in each residual block	100
Filter size	2
Dropout factor	0.02
Maximum epochs	30
Minibatch size	1
Initial learn-rate	0.001
Learn-rate drop factor	0.1
Learn-rate drop period	12 epochs
Gradient threshold	1

LSTM Parameters	
Number of hidden units	600
Dropout factor	0.05
Maximum epochs	30
Minibatch size	1
Initial learn-rate	0.001
Learn-rate drop factor	0.1
Learn-rate drop period	12 epochs
Gradient threshold	1

GRU Parameters	
Number of hidden units	600
Dropout factor	0.05
Maximum epochs	30
Minibatch size	1
Initial learn-rate	0.001
Learn-rate drop factor	0.1
Learn-rate drop period	12 epochs
Gradient threshold	1

Chapter 7

Conclusion & Future Work

In this work, we have proposed CYBEX-P, as a completely automated cybersecurity information sharing (CIS) platform. We have also introduced TAHOE – a graph-based cyberthreat language (CTL), to overcome the limitations of existing CTLs. Moreover, we have introduced a universal Threat Data Query Language (TDQL) to facilitate sharing. Furthermore, we have formulated a novel algorithm called ThreatRank to detect malicious events. We have also tested the scalability and feasibility of CYBEX-P in a real-world setup. Finally, we have shown how to use CYBEX-P infrastructure as a service (IaaS) with two different frameworks. The first one is a phishing URL detection module that uses only one type of threat data namely URLs. The second one is a framework to model attacker behavior in a botnet using heterogeneous threat data. This chapter draws the conclusion for this dissertation, summarises the accomplishments, and discusses possible future works.

7.1 CYBEX-P

Chapter 3 introduces the first contribution of this dissertation: CYbersecurity information Exchange with Privacy (CYBEX-P). CYBEX-P is a robust cybersecurity information sharing (CIS) platform. We have not only designed CYBEX-P but also implemented it in our servers and have tested it with a large amount of threat data.

CYBEX-P can collect threat data from both machines and humans using different channels and protocols. There are numerous sources of threat data with varying network and storage capabilities. We have designed CYBEX-P to interface with all these types of sources. Furthermore, our design ensures that the entire process of data input is completely automated. We have tested our design by setting up five SSH honeypots all over the world. The honeypot locations are Amsterdam, Bangalore, London, Singapore, and Toronto. By continuously collecting a substantial amount of data from these honeypots, we have validated the scalability and reliability of the input module of CYBEX-P. Our implementation and testing show that the CYBEX-P input module can handle a massive influx of data from a variety of sources.

Meanwhile, CYBEX-P uses various standard and custom protocols to communicate with a diverse array of data sources. Therefore, we must ensure a reliable protocol that does not lose data on transit during network or machine failures. We have studied the reliability of the CYEBX-P data collector under various failure and disaster conditions and found it to be highly resilient. Finally, CYBEX-P is developed to potentially handle data streams from thousands of sources. We have optimized all the software

modules, including the input and the API modules, to handle such a large quantity of data. We have achieved this using optimized code, suitable data structures, and well-configured infrastructure. This is tested using heterogeneous data types received from our honeypot sources. Our implementation shows that CYBEX-P can smoothly collect and correlate heterogeneous data from multiple sources.

Concerning correlation, we have developed CYBEX-P not only for mutual data sharing but also for data analysis and advanced correlation. It collects heterogeneous threat data from different types of organizations and correlates them to generate insightful reports. In fact, one of the major strengths of CYBEX-P, which makes it different from previous works, lies in its ability to correlate heterogeneous threat data. Chapter 4 explains the graph-based correlation mechanism facilitated by TAHOE, while chapters 5 and 6 explains the machine learning aspect of it. In particular, chapter 5 describes the framework for a system that uses machine learning to detect phishing URLs. Our system uses CYBEX-P infrastructure as a service (IaaS) to detect previously unseen phishing URLs with an accuracy of 86%. This is a novel contribution of this dissertation which not only explores the horizon of advanced threat data analytics but also demonstrates the capabilities of CYBEX-P for the same. CYBEX-P can facilitate future researchers by automating the data analysis pipeline and making the overall process faster and easier for them.

Another framework that sits on top of CYBEX-P is described in chapter 6. This framework takes a more robust approach towards threat data analysis. A key goal

for CYBEX-P was to correlate heterogeneous threat data and analyze them together. This framework achieves that by modeling attacker behavior in a botnet using four different types of events. The four event types are file downloads, SSH logins, network traffic, and command input into the shell. To the best of our knowledge, no previous work has combined such a diverse dataset to predict one set of targets. We have successfully merged the heterogeneous event types to predict attacker behavior with an accuracy of $85 - 97\%$. Additionally, we have merged datasets from five different locations (Amsterdam, Bangalore, London, Singapore, and Toronto). This framework is a novel contribution that proves the feasibility of CYBEX-P for the correlation and analysis of heterogeneous threat data.

7.2 TAHOE

Chapter 4 introduces the second major contribution of this dissertation: TAHOE a graph-based cyberthreat language (CTL). As previously discussed, any CIS platform like CYBEX-P potentially handles hundreds of different data formats. Thus, it needs a standard data format and structure to represent threat data. A cyberthreat language (CTL) is a specification of how to format and serialize any kind of threat data. CYBEX-P uses TAHOE instead of other CTLs like STIX[35] or MISP core format [36].

Chapter 4 begins with a brief introduction of the TAHOE structure. We have described the data types and corresponding specifications along with the graphical structure that makes TAHOE unique. The core strength of TAHOE lies in the said graphical structure. This structure allows us to intrinsically correlate attributes of different types of events. None of the previous CTLs have that feature rendering them unsuitable for both complex data sharing and advanced data analysis.

We have proven our claim by comparing TAHOE with the two most popular CTLs at present time. Firstly, we have compared TAHOE with STIX in terms of query performance. We have analytically shown that STIX documents are not suitable for indexing in databases. This makes them slow even for simple queries. We have validated this claim using empirical results. Secondly, we have compared TAHOE with the MISP core format. MISP is recently gaining a lot of popularity in the industry for its simple structure and fast queries. The simple structure of MISP is suitable for sharing human-generated reports and observations but fails for complex machine-generated data. We have explained the limitations of the MISP structure using examples in subsection 4.1.3. Furthermore, we have analytically shown how MISP is unsuitable for event simple queries like “fetch all emails from and to a certain email address”.

TAHOE on the other hand was built from the ground up with both complex data formatting and data analysis in mind. TAHOE ob jets can be infinitely nested; in other words, TAHOE can represent arbitrarily complex threat data. Moreover, TAHOE

is optimized for indexing in databases. This makes TAHOE a better alternative to traditional CTLs like STIX or MISP. In addition to these, TAHOE offers several more benefits than traditional CTLs. Firstly, TAHOE normalizes threat data into the same format. Secondly, it is database-independent meaning all the relevant meta-data are stored in a single JSON document. And any container that can store JSON documents can store TAHOE documents. Thirdly, TAHOE de-duplicates data. For example, there can only be one instance of the IP 1.1.1.1 in a TAHOE database. This saves CYBEX-P a lot of storage by not storing the same IP in different events. Fourthly, TAHOE data are globally unique and reproducible. For example, the IP 1.1.1.1 has the same ID in two different TAHOE databases. This ensures TAHOE documents can easily be shared between organizations without any conflict or con-version. Finally, TAHOE edges are bi-directional. This allows us to look up events using attributes or vice-versa. This makes TAHOE robust for complex queries which in turn makes it preferable for complex data analysis.

Another very important contribution of this dissertation is ThreatRank and algorithm that sorts evens based on their maliciousness in a TAHOE database. It does so by correlating newer events with older ones. In simple words, if a new event is assigned a higher ThreatRank it is more likely to be malicious. It should be noted here that the graphical structure of TAHOE enabled us to develop and implement ThreatRank. We have tested the validity of ThreatRank using the famous cyber kill-chain dataset from Lockheed Martin. Our experimental results show that ThreatRank can play a major role in detecting previously unseen malicious events and stopping an attack

before it can cause any damage. We have optimized further optimized ThreatRank algorithm to achieve the best possible complexity performance. Since CYBEX-P will potentially receive an unbounded number of `events` from numerous data sources, ThreatRank needs to be scalable for a large number of `events`. We have validated the performance of this algorithm on a very large dataset with billions of events.

In addition to developing specifications for TAHOE, we have also developed Threat Data Query Language or TDQL. While TAHOE is a specification for how to structure any kind of threat data, TDQL is a specification for how to query any kind of threat data. It standardizes all possible queries in terms of TAHOE instances like events, attributes, objects, and sessions. This would make all kinds of threat data sharing easier for both machines and humans alike.

7.3 Future Work

While we have achieved major goals in this research, plenty of pathways remain for future researchers to extend upon the current endeavors. Firstly, future researchers might be inclined to explore distributed computing paradigms concerning CYBEX-P infrastructure. CYBEX-P will potentially handle a large amount of data with numerous `events`. As a result, it is not possible to handle all the data with one single database. Thus, CYBEX-P is built with a distributed architecture in mind. In turn, ThreatRank also needs to support a distributed architecture to simultaneously

traverse all the parallel databases using distributed computing paradigms like queuing and scheduling.

Secondly, the CYBEX-P system can be used to build more advanced threat data analysis frameworks especially ones that involve neural networks and machine learning. CYBEX-P is particularly suitable for such tasks and made to facilitate such researches. One particular work that is a natural extension of what we have done so far involves labeling or classifying heterogeneous events as benign or malicious. In other words, the methodology developed by us for merging heterogeneous threat data can be applied to all kinds of threat data for classification and anomaly detection. So far we have used only malicious data in the experiments because all of the data obtained from our honeypots are malicious. However, the general procedure still applies for classification tasks involving both benign and malicious data. We will let future researchers explore such a pathway using what we have built so far.

Finally, further research is required on different privacy-preserving queries in CYBEX-P. We believe the more privacy-preserving queries we have the more people will be inclined to adopt CYBEX-P. This is one of the key strengths of CYBEX-P and subsequent research on such queries will make CYBEX-P appealing to a wide range of organizations.

Bibliography

[1] Annual number of data breaches and exposed records in the united states from 2005 to 2020 (in millions). https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/, 2021.

[2] James Lewis. The hidden costs of cybercrime. *Santa Clara: McAfee & CSI (Center for Strategic and International Studies)*, 2020.

[3] Kellen Browing. A 'potentially disastrous' data breach hits twitch, the livestreaming site. *The New York Times*, 2021.

[4] Jessica Silver-Greenberg, Matthew Goldstein, and Nicole Perlroth. Jpmorgan chase hack affects 76 million households. *New York Times*, 2, 2014.

[5] Michael Riley and Jordan Robertson. Jpmorgan hackers said to probe 13 financial firms. *Bloomberg*, 2014.

[6] Devlin Barrett, Emily Glazer, and Kirsten Grind. Jpmorgan hackers said to probe 13 financial firms. *The Wall Street Journal*, 2014.

[7] Xiaokui Shu, Ke Tian, Andrew Ciambrone, and Danfeng Yao. Breaking the target: An analysis of target data breach and lessons learned. *arXiv preprint arXiv:1701.04940*, 2017.

[8] Long Cheng, Fang Liu, and Danfeng Yao. Enterprise data breach: causes, challenges, prevention, and future directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(5):e1211, 2017.

[9] Sam Thielman. Yahoo hack: 1bn accounts compromised by biggest data breach in history. *The Guardian*, 15:2016, 2016.

[10] Selena Larson. Every single yahoo account was hacked-3 billion in all. *CNN business*, 2017.

[11] Lawrence J Trautman and Peter C Ormerod. Corporate directors' and officers' cybersecurity standard of care: The yahoo data breach. *Am. UL Rev.*, 66:1231, 2016.

[12] Rachel Clarke and Taryn Youngstein. Cyberattack on britain's national health service–a wake-up call for modern medicine. *N Engl J Med*, 377(5):409–11, 2017.

[13] James P Farwell and Rafal Rohozinski. Stuxnet and the future of cyber war. *Survival*, 53(1):23–40, 2011.

[14] Daniel Feledi, Stefan Fenz, and Lukas Lechner. Toward web-based information security knowledge sharing. *Information Security Technical Report*, 17(4):199–209, 2013.

[15] S.2588 - Cybersecurity Information Sharing Act of 2014, Congress.gov, Library of Congress., .

[16] S.2717 - Cyber Information Sharing Tax Credit Act of 2014, Congress.gov, Library of Congress., .

[17] Threat Intelligence Platforms. Threatconnect, inc. *URL: http://www.informationweek.com/whitepaper/(access date 21.02.2017).*

[18] AlienVault. The open source siem. *URL: https://www.alienvault.com-/products/ossim (accessed 25.06.2017).*

[19] IBM. X-force. *URL: https://www.ibm.com/security/xforce (accessed 13.04.2020).*

[20] Anomali. Threatstream. *URL: https://www.anomali.com/products/-threatstream (accessed 13.04.2020).*

[21] Facebook Inc. Threatexchange. *URL: https://developers.facebook.com/-programs/threatexchange (accessed 13.04.2020).*

[22] Eclecticiq. *URL: https://www.eclecticiq.com/ (accessed 13.04.2020).*

[23] Julie Connolly, Mark Davidson, and Charles Schmidt. The trusted automated exchange of indicator information (TAXII). *The MITRE Corp.*, pages 1–20, 2014.

[24] Cynthia Wagner, Alexandre Dulaunoy, Gérard Wagener, and Andras Iklody. Misp: The design and implementation of a collaborative threat intelligence sharing platform. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*, pages 49–56. ACM, 2016.

[25] Diego Fernández Vázquez, Oscar Pastor Acosta, Christopher Spirito, Sarah Brown, and Emily Reid. Conceptual framework for cyber defense information sharing within trust relationships. In *2012 4th International Conference on Cyber Conflict (CYCON 2012)*, pages 1–17. IEEE, 2012.

[26] Farzan Kolini and Lech Janczewski. Exploring incentives and challenges for cybersecurity intelligence sharing (cis) across organizations: A systematic review.

[27] Luc Dandurand and Oscar Serrano Serrano. Towards improved cyber security information sharing. In *2013 5th International Conference on Cyber Conflict (CYCON 2013)*, pages 1–16. IEEE, 2013.

[28] Clemens Sauerwein, Christian Sillaber, Andrea Mussmann, and Ruth Breu. Threat intelligence sharing platforms: An exploratory study of software vendors and research perspectives. 2017.

[29] Christian Sillaber, Clemens Sauerwein, Andrea Mussmann, and Ruth Breu. Data quality challenges and future research directions in threat intelligence sharing practice. In *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*, pages 65–70, 2016.

[30] Rebekah Brown and Robert M Lee. The evolution of cyber threat intelligence (cti): 2019 sans cti survey. *SANS Institute. Available online: https://www. sans. org/white-papers/38790/(accessed on 12 July 2021)*, 2019.

[31] Piotr Kijewski and Paweł Pawliński. Proactive detection and automated exchange of network security incidents. *Abgerufen am*, 20, 2014.

[32] Waleed Alkalabi, Leonie Simpson, and Hasmukh Morarji. Barriers and incentives to cybersecurity threat information sharing in developing countries: A case study of saudi arabia. In *2021 Australasian Computer Science Week Multiconference*, pages 1–8, 2021.

[33] Konstantinos Rantos, Arnolnt Spyros, Alexandros Papanikolaou, Antonios Kritsas, Christos Ilioudis, and Vasilios Katos. Interoperability challenges in the cybersecurity information sharing ecosystem. *Computers*, 9(1):18, 2020.

[34] B McConnell. and the department of homeland security, enabling distributed security in cyberspace: Building a healthy and resilient cyber ecosystem with automated collective action, 2011.

[35] Sean Barnum. Standardizing cyber threat intelligence information with the structured threat information expression (stix). *MITRE Corporation*, 11:1–22, 2012.

[36] Alexandre Dulaunoy and Andras Iklody. MISP core format. Internet-Draft draft-dulaunoy-misp-core-format-13, Internet Engineering Task Force, December 2020. URL `https://datatracker.ietf.org/doc/html/draft-dulaunoy-misp-core-format-13`. Work in Progress.

[37] Charles Edwards, Samuel Migues, Roger Nebel, and Daniel Owen. System and method of data collection, processing, analysis, and annotation for monitoring cyber-threats and the notification thereof to subscribers, March 28 2002. US Patent App. 09/950,820.

[38] Wanying Zhao and Gregory White. A collaborative information sharing framework for community cyber security. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, pages 457–462. IEEE, 2012.

[39] George D Webster, Zachary D Hanif, Andre LP Ludwig, Tamas K Lengyel, Apostolis Zarras, and Claudia Eckert. Skald: a scalable architecture for feature extraction, multi-user analysis, and real-time information sharing. In *International Conference on Information Security*, pages 231–249. Springer, 2016.

[40] Lawrence A Gordon, Martin P Loeb, and William Lucyshyn. Sharing information on computer systems security: An economic analysis. *Journal of Accounting and Public Policy*, 22(6):461–485, 2003.

[41] Lawrence A Gordon, Martin P Loeb, William Lucyshyn, and Lei Zhou. The impact of information sharing on cybersecurity underinvestment: A real options perspective. *Journal of Accounting and Public Policy*, 34(5):509–519, 2015.

[42] Jorge L Hernandez-Ardieta, Juan E Tapiador, and Guillermo Suarez-Tangil. Information sharing models for cooperative cyber defence. In *Cyber Conflict (CyCon), 2013 5th International Conference on*, pages 1–28. IEEE, 2013.

[43] Farhan Sadique, Sui Cheung, Iman Vakilinia, Shahriar Badsha, and Shamik Sengupta. Automated structured threat information expression (stix) document generation with privacy preservation. In *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)(IEEE UEMCON 2018)*, 2018.

[44] Farhan Sadique, Khalid Bakhshaliyev, Jeff Springer, and Shamik Sengupta. A system architecture of cybersecurity information exchange with privacy (cybexp). In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0493–0498. IEEE, 2019.

[45] Standards and tools for exchange and processing of actionable information. `https://www.enisa.europa.eu/publications/standards-and-tools-for-exchange-and-processing-of-actionable-information/`.

[46] Jessica Steinberger, Anna Sperotto, Mario Golling, and Harald Baier. How to exchange security events? overview and evaluation of formats and protocols. In *Integrated Network Management (IM), 2015 IFIP/IEEE International*

Symposium on, pages 261–269. IEEE, 2015.

[47] Panos Kampanakis. Security automation and threat information-sharing options. *IEEE Security & Privacy*, 12(5):42–51, 2014.

[48] Deepak K Tosh, Shamik Sengupta, Sankar Mukhopadhyay, Charles A Kamhoua, and Kevin A Kwiat. Game theoretic modeling to enforce security information sharing among firms. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*, pages 7–12. IEEE, 2015.

[49] Deepak Tosh, Shamik Sengupta, Charles Kamhoua, Kevin Kwiat, and Andrew Martin. An evolutionary game-theoretic framework for cyber-threat information sharing. In *Communications (ICC), 2015 IEEE International Conference on*, pages 7341–7346. IEEE, 2015.

[50] Charles Kamhoua, Andrew Martin, Deepak K Tosh, Kevin A Kwiat, Chad Heitzenrater, and Shamik Sengupta. Cyber-threats information sharing in cloud computing: A game theoretic approach. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, pages 382–389. IEEE, 2015.

[51] Iman Vakilinia and Shamik Sengupta. A coalitional game theory approach for cybersecurity information sharing. In *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*, pages 237–242. IEEE, 2017.

[52] Mark E DeYoung, Philip Kobezak, David Raymond, Randy Marchany, and Joseph Tront. Privacy preserving network security data analytics: Architectures and system design. In *51st Hawaii International Conference on System Sciences, 2018*. University of Hawaii at Manoa, 2018.

[53] Vicka Corey, Charles Peterman, Sybil Shearin, Michael S Greenberg, and James Van Bokkelen. Network forensics analysis. *IEEE Internet Computing*, 6(6):60–66, 2002.

[54] Alec Yasinsac and Yanet Manzano. Policies to enhance computer and network forensics. In *Proceedings of the 2001 IEEE workshop on information assurance and security*, pages 289–295, 2001.

[55] Emmanuel S Pilli, RC Joshi, and Rajdeep Niyogi. A generic framework for network forensics. *International Journal of Computer Applications*, 1(11):1–6, 2010.

[56] Kulesh Shanmugasundaram, Nasir Memon, Anubhav Savant, and Herve Bronnimann. Fornet: A distributed forensics network. In *International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 1–16. Springer, 2003.

[57] MI Cohen. Pyflag–an advanced network forensic framework. *Digital investigation*, 5:S112–S120, 2008.

[58] Yechiam Yemini. The osi network management model. *IEEE Communications Surveys & Tutorials*, 3(1):20–29, 1993.

[59] Wei Wang and Thomas E Daniels. A graph based approach toward network forensics analysis. *ACM Transactions on Information and System Security (TISSEC)*, 12(1):4, 2008.

[60] Emmanuel S Pilli, Ramesh C Joshi, and Rajdeep Niyogi. Network forensic frameworks: Survey and research challenges. *digital investigation*, 7(1-2):14–27, 2010.

[61] Natarajan Meghanathan, Sumanth Reddy Allam, and Loretta A Moore. Tools and techniques for network forensics. *arXiv preprint arXiv:1004.0570*, 2010.

[62] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications*, 66:214–235, 2016.

[63] Ahmad Almulhem. Network forensics: Notions and challenges. In *2009 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 463–466. IEEE, 2009.

[64] Srinivas Mukkamala and Andrew H Sung. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of digital evidence*, 1(4):1–17, 2003.

[65] Niandong Liao, Shengfeng Tian, and Tinghua Wang. Network forensics based on fuzzy logic and expert system. *Computer Communications*, 32(17):1881–1892, 2009.

[66] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2. 2008.

[67] Farhan Sadique. Tahoe. `https://github.com/cybex-p/tahoe`, 2021.

[68] Kyle Banker. *MongoDB in action*. Manning Publications Co., 2011.

[69] Michel Oosterhof. Cowrie honeypot. *Security Intelligence*, 2014.

[70] Phishtank. `http://www.phishtank.com`. Accessed: 2019-10-30.

[71] OpenPhish. `http://www.openphish.com`. Accessed: 2019-11-22.

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

[73] Farhan Sadique, Raghav Kaul, Shahriar Badsha, and Shamik Sengupta. An automated framework for real-time phishing url detection. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0335–0341. IEEE, 2020.

[74] The ponemon 2021 cost of phishing study. `https://www.proofpoint.com/us/resources/analyst-reports/ponemon-cost-of-phishing-study`. Accessed: 2021-11-06.

[75] Tom N Jagatic, Nathaniel A Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *Communications of the ACM*, 50(10):94–100, 2007.

[76] Enterprise Phishing Resiliency and Defense Report, 2017.

[77] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. Malicious url detection using machine learning: a survey. *arXiv preprint arXiv:1701.07179*, 2017.

[78] Quarterly Threat Trends. `https://www.webroot.com/us/en/business/-resources/threat-trends/june-2019/`, 2019. [Online; accessed 1-September-2019].

[79] Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. Urlnet: learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*, 2018.

[80] Ya-Lin Zhang, Longfei Li, Jun Zhou, Xiaolong Li, Yujiang Liu, Yuanchao Zhang, and Zhi-Hua Zhou. Poster: A pu learning based system for potential malicious url detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2599–2601. ACM, 2017.

[81] Rakesh Verma and Avisha Das. What's in a url: Fast feature extraction and malicious url detection. In *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*, pages 55–63. ACM, 2017.

[82] Jianguo Jiang, Jiuming Chen, Kim-Kwang Raymond Choo, Chao Liu, Kunying Liu, Min Yu, and Yongjian Wang. A deep learning based online malicious url and dns detection scheme. In *International Conference on Security and Privacy in Communication Systems*, pages 438–448. Springer, 2017.

[83] Annabella Astorino, Antonino Chiarello, Manlio Gaudioso, and Antonio Piccolo. Malicious url detection via spherical classification. *Neural Computing and Applications*, 28(1):699–705, 2017.

[84] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. Lexical feature based phishing url detection using online learning. In *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, pages 54–60. ACM, 2010.

[85] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254. ACM, 2009.

[86] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):21, 2011.

[87] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Learning to detect malicious urls. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):30, 2011.

[88] Guolin Tan, Peng Zhang, Qingyun Liu, Xinran Liu, Chunge Zhu, and Li Guo. Malfilter: A lightweight real-time malicious url filtering system in large-scale networks. In *2018 IEEE ISPA/IUCC/BDCloud/SocialCom/SustainCom*, pages 565–571. IEEE, 2018.

[89] Ozgur Koray Sahingoz, Ebubekir Buber, Onder Demir, and Banu Diri. Machine learning based phishing detection from urls. *Expert Systems with Applications*, 117:345–357, 2019.

[90] Mohammad Saiful Islam Mamun, Mohammad Ahmad Rathore, Arash Habibi Lashkari, Natalia Stakhanova, and Ali A Ghorbani. Detecting malicious urls using lexical analysis. In *International Conference on Network and System Security*, pages 467–482. Springer, 2016.

[91] Scott Hollenbeck and Andrew Newton. Registration data access protocol (rdap) query format. RFC 7842, IETF, March 2015. URL `https://tools.ietf.org/html/rfc7482`.

[92] GeoLite2 Free Downloadable Databases. `https://dev.maxmind.com/geoip/geoip2/geolite2/`. [Online; accessed 19-September-2019].

[93] Terry Anderson. *The theory and practice of online learning*. Athabasca University Press, 2008.

[94] Yue Wu, SC Hoi, and Nenghai Yu. Libsol: A library for scalable online learning algorithms. *SMU Technical Report (SMU-TR-2016-07-25)*, 2016.

[95] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[96] Ken Dunham and Jim Melnick. *Malicious bots: an inside look into the cyber-criminal underground of the internet*. CrC Press, 2008.

[97] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th {USENIX$} $ security symposium (${$USENIX$}$ Security 17)*, pages 1093–1110, 2017.

[98] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647, 2009.

[99] Seungwon Shin and Guofei Gu. Conficker and beyond: a large-scale empirical study. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 151–160, 2010.

[100] Ramneek Puri. Bots & botnet: An overview. *SANS Institute*, 3, 2003.

[101] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *2009 3rd International Conference on Emerging Security Information, Systems and Technologies*. IEEE, 2009.

[102] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[103] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[104] Ahmad Karim, Rosli Bin Salleh, Muhammad Shiraz, Syed Adeel Ali Shah, Irfan Awan, and Nor Badrul Anuar. Botnet detection techniques: review, future trends, and issues. *Journal of Zhejiang University SCIENCE C*, 15(11):943–983, 2014.

[105] Anestis Karasaridis, Brian Rexroad, David A Hoeflin, et al. Wide-scale botnet detection and characterization. *HotBots*, 7:7–7, 2007.

[106] James R Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection. *SRUTI*, 6:7–7, 2006.

[107] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. 2008.

[108] Hyunsang Choi, Hanwoo Lee, Heejo Lee, and Hyogon Kim. Botnet detection by monitoring group activities in dns traffic. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*. IEEE, 2007.

[109] Ricardo Villamarín-Salomón and José Carlos Brustoloni. Identifying botnets using anomaly detection techniques applied to dns traffic. In *2008 5th IEEE Consumer Communications and Networking Conference*. IEEE, 2008.

[110] David Dagon. Botnet detection and response. In *OARC workshop*, 2005.

[111] Antoine Schonewille and Dirk-Jan Van Helmond. The domain name service as an ids. *Research Project for the Master System-and Network Engineering at the University of Amsterdam*, 2006.

[112] S Murugan and K Kuppusamy. System and methodology for unknown malware attack. 2011.

[113] Gideon Creech and Jiankun Hu. A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns. *IEEE Transactions on Computers*, 63(4):807–819, 2013.

[114] Linqiang Ge, Hui Liu, Difan Zhang, Wei Yu, Rommie Hardy, and Robert Reschly. On effective sampling techniques for host-based intrusion detection

in manet. In *MILCOM 2012-2012 IEEE Military Communications Conference*, pages 1–6. IEEE, 2012.

[115] Farhan Sadique, Ignacio Astaburuaga, Raghav Kaul, Shamik Sengupta, Shahriar Badsha, James Schnebly, Adam Cassell, Jeff Springer, Nancy Latourrette, and Sergiu M. Dascalu. Cybersecurity information exchange with privacy (cybex-p) and tahoe – a cyberthreat language, 2021.

[116] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[117] Lance Spitzner. The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 1(2):15–23, 2003.

[118] Michel Oosterhof. Cowrie ssh/telnet honeypot, 2016.

[119] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[120] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[121] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

[122] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[123] MATLAB. *version 9.10.0.1602886 (R2021a)*. The MathWorks Inc., Natick, Massachusetts, 2021.