IMPROVING CYBER SITUATIONAL UNDERSTANDING

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

By

Philip Huff
Harding University
Bachelor of Science in Mathematics and Computer Science, 2002
James Madison University
Master of Science in Computer Science, 2008 July 2021

University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

Qinghua Li, Ph.D.

Dissertation Advisor and Committee Chair

Susan Gauch, Ph.D.

Committee Member

Roy A. McCann, Ph.D.

Committee Member

Brajendra Nath Panda, Ph.D.

Committee Member

Abstract

Effective cybersecurity operations require the ability to analyze large amounts of information to assess security risks and formulate defensive strategies against adversaries. This has become more complex in recent years as the sprawl and interconnectivity of devices grows through implementation of virtualization, cloud computing, and Internet of Things (IoT). The amount of data and analysis required for effective cybersecurity command and control decisions far exceeds humans' capacity to perform manually. We characterize the analysis problem as cyber situational understanding. The research presented to improve cyber situational understanding focuses on vulnerability analysis and threat intelligence.

Regarding vulnerabilities, entities must analyze and plan work for between thousands and tens of thousands of software vulnerabilities annually. Entities heavily use network firewalls to limit vulnerability exposure. As a result, some of these vulnerabilities permit exposure to adversarial exploitation, whereas others are inaccessible and therefore present negligible risk of exploitation. Distinguishing between high and low risk software vulnerabilities requires a deep understanding of the vulnerability, network firewall protection, and characteristics of the targeted device. This problem is solved by extracting network service features from vulnerability data features using both machine-learning and natural language processing. Then, the network firewall topology is parsed to determine which vulnerabilities are reachable by adversaries. Ultimately, a state-based safety analysis ascertains which vulnerabilities are unsafe.

A related vulnerability analysis problem occurs in cybersecurity operations when associating an entity's hardware and software assets to public vulnerability databases. Assets often reveal hardware and software through installation artifacts and network service identification, and entities store these artifacts in inventory databases. However, software and hardware vendors apply a standard Common Platform Enumeration (CPE) naming convention when publicly reporting vulnerabilities. Associating these two datasets often requires many hours to days of manual inspection. The proposed solution automates the mapping approach of human analysts using fuzzy matching techniques, natural language processing, and, ultimately, machine learning to present

a small set of recommendations for mapping the two datasets. The result significantly reduces human analysis time and reduces the occurrence of false positives in vulnerability notifications.

Finally, cyber threat intelligence (CTI) requires associating cyber observable artifacts, such as IP addresses, URIs, and file hashes, with cyber threat tactics, techniques, and procedures. Unfortunately, most CTI data is compartmentalized across multiple organizations and cannot be shared due to the legal and reputational risk with cyber threat being associated with the entity. The approach to solving this problem inovlves using a distributed ledger with anonymous token spending and authentication. This allows a consortium of semi-trusted entities to share the workload of curating CTI for a threat sharing community's cooperative benefit.

Acknowledgements

I wish to express my deepest gratitude to my advisor, Dr. Qinghua Li, for his support, encouragement, and willingness to guide me in transitioning from industry to academia.

I appreciate my committee members Drs. Susan Gauch, Roy McCann, and Brajendra Nath Panda for their time and support, and valuable suggestions for developing this dissertation.

Thank you to my research partners Matthew Kennett, Kylie McClanahan, and Fengli Zhang with whom I am grateful to have shared time in your creative process. Also, I wish to thank my colleagues from the University of Arkansas at Little Rock: Drs. Carolina Cruz, Dirk Rieners, Jan Springer, Al Baker, Sandra Leiterman, and many more who have allowed me to complete this degree.

To my wife, Joy, and children, Joanna, Abigail, Mac, and Kyle, I could not imagine any endeavor without your support and devotion.

And above all, I thank God for inspiring and sustaining me.

This dissertation is supported in part by the DOE under awards DE-OE0000779 and DE-CR0000003, and by the NSF under award number 1751255.

Table of Contents

List of Figures

List of Tables

Published Papers

- **Chapter 2: Philip Huff**, Qinghua Li. "Towards Automated Assessment of Vulnerability Exposures in Security Operations". In: *EAI International Conference on Security and Privacy in Communication Networks, 2021*. (Accepted)
- **Chapter 3: Philip Huff**, Qinghua Li. "A Distributed Ledger for Non-Attributable Cyber Threat Intelligence Exchange". In: *EAI International Conference on Security and Privacy in Communication Networks, 2021*. (Accepted)
- **Chapter 4: Philip Huff**, Kylie McClanahan, Thao Le, and Qinghua Li. "A Recommender System for Tracking Vulnerabilities". In: *International Workshop on Next Generation Security Operations Centers (NG-SOC), 2021*. (Accepted)

1 Introduction

Cybersecurity operations encompass the tactical decision-making process to protect computer systems in response to dynamic adversarial threats. Quality decision-making is increasingly challenging with the growth of complexity in both the computing environment and adversarial schemes. Reliance on repetitive human analysis limits the advancement and scalability of cybersecurity operations.

Borrowing military terminology, applying analysis and judgment to presented information, and determining the relationship among operational variables is known as situation understanding [7]. More specifically, for cybersecurity operations, situation understanding means obtaining the knowledge necessary to protect the cyber system effectively.

The number of devices and dense coupling of system components make cyber situation understanding difficult to obtain. Trends in virtualization, cloud computing, and Internet-of-Things deployment contribute to an analysis task growing beyond humans' capacity to perform manually. This creates the following types of uncertainty in cybersecurity operations and attack paths:

1. **Interconnectedness** - Devices establish communication channels cheaply through ubiquitous networks and transparently through virtualized networking

2. **Software Proliferation** - A single device comprises several hundred software components, many of which have a further tree of software dependencies

3. **Vulnerabilities** - Each software component regularly has common weaknesses identified and publicly reported

4. **Adversarial Behavior** - Changing adversarial tactics, techniques, and procedures (TTP) leaves behind trace indicators (e.g., IP addresses, URIs, file hashes, etc.), which assist in detecting and preventing future attacks. However, the rate at which indicators are produced has an overwhelming effect on cybersecurity operations.

In one sense, cyber situational understanding is a big data problem, but the data is compartmentalized across organizations. The legal, regulatory, and reputational trust barriers minimize the possibility of aggregation. Consequently, although cyber situational understanding has a critical need of machine learning over big data, the possibility is currently unrealized.

The contributions of this research solve problems in cyber situational understanding through both (i) existing public datasets and (ii) the creation of a new semi-public dataset on which machine learning can be performed. The most consistent public dataset for cybersecurity operations exists with the National Vulnerability Database (NVD) [64].

This dissertation research is organized into three parts. The first part tackles the problem of cyber situation understanding in software vulnerabilities by defining a safe state assessment and automatically performing the assessment over many vulnerabilities. The second part addresses the problem of associating an entity's hardware and software inventory to public vulnerability repositories. The association allows entities to more fully automate the vulnerability assessment while avoiding false negatives and reducing false positive vulnerability notifications. Then part three explores the cyber threat intelligence (CTI) analysis problem using a distributed ledger. The data sharing barriers are addressed using a new approach to anonymous spending and authentication on a permissioned blockchain, thereby enabling faster curation of CTI. Each problem and the contributions are summarized below.

1.1 Automating the Assessment of Vulnerability Exposures [46]

In chapter 2, cyber situation understanding for software vulnerabilities is improved through an automated state-based safety assessment. Current approaches for risk analysis of software vulnerabilities using manual assessment and numeric scoring do not complete fast enough to keep pace with the maintenance work rate to patch and mitigate the vulnerabilities. This chapter proposes a new approach to modeling software vulnerability risk in the context of the network environment and firewall configuration. In the approach, vulnerability features are automatically matched up with networking, target asset, and adversary features to determine whether adversaries can exploit

a vulnerability. The ability of adversaries to reach a vulnerability is modeled by automatically identifying the network services associated with vulnerabilities through a pipeline of machine learning and natural language processing and automatically analyzing network reachability. Our results show that the pipeline can identify network services accurately. We also find that only a small number of vulnerabilities pose real risks to a system. However, if left unmitigated, adversarial reach to vulnerabilities may extend to nullify the effect of firewall countermeasures.

1.2 A Recommender System for Tracking Vulnerabilities [47]

In chapter 3, mitigating vulnerabilities in software requires first identifying the vulnerabilities with an organization's software assets. This seemingly trivial task involves maintaining vendor product vulnerability notifications for a kludge of hardware and software packages from innumerable software publishers, coding projects, and third-party package managers. On the other hand, software vulnerability databases are often consistently reported and categorized in clean, standard formats and neatly tied to a common software platform enumerator (i.e., CPE). Currently it is a heavy workload for cybersecurity analysts to match their hardware and software package inventory to target CPEs. This hinders organizations from getting notifications for new vulnerabilities, and identifying applicable vulnerabilities. In this chapter, we present a recommender system to automatically identify a minimal candidate set of CPEs for software names to improve vulnerability identification and alerting accuracy. The recommender system uses a pipeline of natural language processing, fuzzy matching, and machine learning to significantly reduce the human effort needed for software product vulnerability matching.

1.3 Cyber Threat Intelligence Exchange [45]

Finally, in chapter 4 presents an approach to improve the cyber situation understanding of cyber threats. Cyber threat intelligence (CTI) sharing provides cybersecurity operations an advantage over adversaries by more quickly characterizing the threat, understanding its tactics, and anticipating the objective. However, organizations struggle with sharing threat intelligence due, in part,

due to the legal and financial risk of being associated with a potential malware campaign or threat group. An entity wishing to share threat information or obtain information about a specific threat risks being associated with the threat actors, resulting in costly legal disputes, regulatory investigation, and reputational damage. As a result, the threat intelligence data needed for cybersecurity situational awareness often lacks in volume, quality, and timeliness. We propose a distributed blockchain ledger to facilitate sharing cybersecurity threat information and provide a mechanism for entities to have non-attributable participation in a threat-sharing community. Learning from approaches to Distributed Anonymous Payment (DAP) schemes in cryptocurrency, we use a new token-based authentication scheme for use in a permissioned blockchain. This allows a consortium of semi-trusted entities to share the workload of curating CTI for the community's cooperative benefit.

1.4 Summary of Contributions

My contributions are summarized as follows:

- A formal definition of system state safety when combining vulnerability, adversary, and target asset features, and an automation framework for assessing the system security, which includes data modeling, extraction of network service information from vulnerability features/descriptions, network reachability analysis under firewall rules, and model checking vulnerability safety. The reachability analysis is enabled through an artificial intelligence pipeline including ML and NLP methods to identify the network services associated with vulnerabilities based on vulnerability features and descriptions enables automating the association between vulnerabilities and firewall policies.

- Matching hardware and software inventories to public vulnerability repositories through a pipeline of natural language processing, fuzzy matching, and machine learning.

- Provide a solution for entities to share observed CTI without attribution using a permissioned blockchain. We propose a novel approach to a Distributed Anonymous Payment

(DAP) scheme for permissioned blockchains to allow for anonymous transactions in CTI sharing. The solution also efficiently maintains anonymous authentication in CTI sharing and provide revocation services for entities by splitting maintenance of the Merkle tree used for anonymous authentication between participating peers, which allows for more regular updates of the Merkle Tree across the distributed ledger.

References

[7] *Army Doctrine Publication No. 6.0: Mission Command, Command and Control of Army Forces*. Department of the Army Headquarters. July 2019.

[45] Philip Huff and Qinghua Li. "A Distributed Ledger for Non-Attributable Cyber Threat Intelligence Exchange". In: *EAI International Conference on Security and Privacy in Communication Networks*. 2021.

[46] Philip Huff and Qinghua Li. "Towards Automated Assessment of Vulnerability Exposures in Security Operations". In: *EAI International Conference on Security and Privacy in Communication Networks*. 2021.

[47] Philip Huff et al. "A Recommender System for Tracking Vulnerabilities". In: *International Workshop on Next Generation Security Operations Centers (NG-SOC)*. 2021.

[64] *National Vulnerability Database Data Feed*. https://nvd.nist.gov/vuln/data-feeds. Accessed: 2020-01-28.

2 Automating the Assessment of Vulnerability Exposures

The actual number of software vulnerabilities has become more evident with bug bounty programs, automated code analysis, and increased reporting by software vendors. In 2017, the number of vulnerabilities reported annually through the National Vulnerability Database (NVD) doubled and currently continues an upward trend [65]. Vulnerability mitigation for servers and other autonomous devices requires extensive planning, coordination, and testing. Consequently, the burden to maintain secure operations in organizations often exceeds the available resources.

To address this problem, defenders in an organization need a more contextual understanding of the actual risk posed by a vulnerability. Contextual risk assessment requires understanding (i) an adversary's tactics, capabilities, and access to a targeted vulnerability and (ii) the effectiveness of existing mitigation in the organization. Moreover, defenders need the risk information quickly. For example, in a 2017 Equifax breach, a months-old unpatched Apache Struts vulnerability was identified as the initial attack vector [22]. If the degree of risk became evident upon release of the vulnerability, operators could have immediately patched the software.

A commonly used defense is a firewall. Thus, one promising solution for providing contextual risk information to operators in determining whether an adversary has the needed network access to exploit given vulnerabilities under firewall rules. The challenge is mapping the many applicable software vulnerabilities of a system to the firewall rules. Currently, operators can only perform this manually.

Our work bridges this gap by automating the identification of network services used to exploit vulnerabilities through a pipeline of machine learning (ML) and natural language processing (NLP) methods. The machine learning method uses standard vulnerability features from the NVD data feed to predict the associated network service. The NLP method further boosts the overall prediction accuracy with information from vulnerability descriptions. Experiments show that the pipeline can identify network services for 97% of vulnerabilities with an accuracy of 95%.

The joining of firewall and vulnerability data allows identifying which vulnerabilities are accessible outside of their segmented network zone. It then becomes possible to model an adversary's

external view of the vulnerability. To do this, we model the placement of adversaries in the Internet and enterprise network zones and develop methods for network reachability analysis under firewall rules.

Once the adversary's ability to reach the vulnerability is determined, the system's security state can be precisely assessed. Using standard features for access, capability, and impact in the Common Vulnerability Scoring System (CVSS), we model safety as a function of set dominance between the vulnerability, target asset, and adversary.

The approach demonstrates that over a realistic sample system only a small portion of vulnerabilities are unsafe. The practical result signifies a reduced effort for the defender to maintain a system's secure state. The model can also recursively iterate to show how adversaries might extend their reach using already reached software vulnerabilities. We refer to vulnerabilities in this path as gateway vulnerabilities and demonstrate the detriment they may have on the entire system's safety.

Section 4.1 reviews related work. Section 2.2 introduces data modeling. Section 2.3 describes how to identify the network service associated with vulnerabilities. Section 2.4 presents network reachability analysis under firewall rules. Section 2.5 presents the safety model for vulnerability exposure checking. The last two sections of this chapter present evaluation results and conclusions.

2.1 Related Work

The concept of assessing software vulnerability risk in terms of adversarial capability has its roots in the broader field of attack trees. Attack Trees, initially pioneered by Schneier [76], are practical and well-established modeling tools for automatically assessing risk by refining the ultimate goal of an attacker into a granular tree of actions to quantify the risk of an attack. Later research provides a formal specification for attack trees [59]. Attack-Defense Trees (AD-Trees) add the analysis of defense mitigation in the presence of attack methodologies to assess both mitigation approaches and risk of attack [50]. Recent solutions in automated AD-Tree generation [8], multi-parameter risk optimization [36] and automatically relating attacks to attack tree goals [58] continue to propel AD-Trees as a practical tool to optimize vulnerability mitigation. Our approach differs from AD-

Trees by focusing only on software vulnerabilities from the perspective of a defender. In assessing software vulnerabilities, we model the simple attacker goal to exploit the system and use standard atomic attributes to measure the attacker's capability.

Network attack graphs have similar objectives to attack trees in identifying adversarial capability to attack but focus on the target reachability by the attacker. The use of modeling the physical network as a graph to assess an attacker's capability to exploit vulnerabilities originated in work [70] and [4]. In [88], they provide a grammar for defining connectivity in a network and propose a model-checking safety invariant for assessing vulnerabilities. This approach is expanded in [86] to include a more general safety condition against unknown or zero-day attacks.

Several papers have suggested approaches to automating the software vulnerability assessment using network attack graphs. In [90, 40], they propose metrics for a qualitative security score based on vulnerabilities present in the network. Similarly, [67] combines vulnerability metric data with firewall topology to provide an overall view of risk using various metrics, including connectivity and length of network paths. A more recent approach involves scoring network path edges using applicable vulnerability metrics to host data to calculate risk as a function of the path cost [38].

AD-Trees and attack graphs have the same nuisance of overwhelming the security analyst with risk metrics and attack scenarios. Our approach overcomes this obstacle by focusing more narrowly on the common problem of software vulnerability management using standard data features (i.e., CVSS) well understood by practitioners. Instead of outputting a graph or risk score which still needs much manual analysis to decide whether vulnerabilities need mitigation or not, our model generates a deterministic output as to whether vulnerabilities are safe from attackers or not. We abstract much of the complexity in decision making using set dominance similar to other areas of formal models in computer security such as access control [53] and, more recently, in trusted computing [91]. Also, existing work does not address the automated extraction of network services from vulnerability features and descriptions.

Some studies have used the NVD data for security purposes. [55] uses NLP over vulnerability descriptions for extracting new entities (i.e., Named-Entity Recognition or NER) to generally

describe vulnerabilities in terms of cause, consequence analysis, and impact estimation. [87] uses ML models for attack classification and improved impact scoring, and [56] uses concept drift in NLP to assess vulnerabilities based on their descriptions. [96], and [94] use machine learning to recommend remediation actions for and predict the probabilistic risk levels of vulnerabilities, and [60] uses natural language processing over vulnerability descriptions to identify mitigation information. [47] studies how to map software assets to to vulnerabilities. See [84] for a repository of work in this domain. However, these existing studies do not automatically extract network services from vulnerability features and descriptions, and they do not consider firewall policies as our work does.

2.2 Data Modeling

Our safety model seeks to understand whether and how a set of adversaries can exploit a given vulnerability. This section describes the relevant data for understanding adversarial interaction. A significant portion of the input data comes from the NVD as distinct attributes, which provides a consistent and timely source for real-time vulnerability analysis.

2.2.1 Vulnerability Features, Asset Features, and Adversary Capabilities

The NVD provides a full data feed of twelve attributes associated with the CVSS. CVSS is an open standard maintained by a special interest group under the Forum of Incident Response and Security Teams (FIRST) [26]. Software publishers broadly use it to describe security vulnerabilities in their software.

Here, we describe the attributes related to the adversarial capability necessary to exploit vulnerabilities as a function of state labeling propositions that we use for modeling. Each feature labels a distinct capability, representing a cumulative set hierarchy for deterministically calculating adversarial interaction requirements.

Features in the NVD have an abbreviation convention, which we conveniently adopt with state labeling. The Attack Vector, AV, label defines the access necessary for an exploit. The

propositions *Physical* (P), *Local* (L), *Network* (N) and *Adjacent* (A) are an ordered set $AV = \{N, A, L, P\}$ in terms of decreasing exploit opportunity with respect to the vulnerability and increasing exploit difficulty for the adversary. An attack vector of N implies that the adversary can exploit the vulnerability directly through a network service. In contrast, an attack vector of L implies the adversary needs to interact with the device for exploitation. Local attacks do not necessarily mean an adversary cannot perform the attack remotely. For example, an adversary can interact through VNC or SSH to exploit a vulnerability with an attack vector of L.

Attack Complexity, AC, describes the difficulty required to develop an exploit for a given vulnerability. Propositions include *Low*, L, and *High*, H, with the ordered set $AC = \{L, H\}$. For example, low attack complexity would indicate an adversary's greater opportunity to exploit the vulnerability.

Privileges, PR, describes the level of privileges necessary to exploit the vulnerability and is similar to AC with the additional possibility of no privileges, N required. Thus, the ordered set would be $PR = \{N, L, H\}$ in terms of decreasing opportunity for exploitation.

The User Interaction label, UI, indicates the degree to which a human must be involved to exploit the vulnerability. Propositions include *Required*, R, and *None*, N, with the ordered set as $UI = \{R, N\}$. When R applies to a device, it would indicate regular user interaction and have more exploit opportunities.

The temporal metric of exploitability, EX, describes the current availability of code to exploit a vulnerability. The label EX propositions include *High*, H, meaning exploit code is widely available, *Functional*, F, meaning exploit code is available but may require additional work, *Proof-of-concept*, P, and *Unproven*, U, where the exploit code is not known to be developed. The ordered set is $EX = \{H, F, P, U\}$ with decreasing exploitability of a vulnerability.

Although the CVSS attributes describe vulnerability features, we make a key observation that these features apply to both (i) target assets associated with the vulnerability and (ii) a prospective adversary's capability. Table 1 describes the relationship of the CVSS capability features. We capitalize on these relationships in section 2.5 to precisely define capability in terms of safety.

Table 1: CVSS-Based Data Features

CVSS Feature	***Vulnerability***	***Adversary***	***Asset***
Attack Vector	The physical or network access for exploit	The ability of an adversary to use the path	The location of an asset
Privileges	The logical access necessary for exploit	The level of privileges available to the adversary	
User Interaction	Whether an exploit needs interaction with a human		Whether humans interact on the asset
Exploitability	The availability and ease of developing exploit code	The ability of an adversary to use or develop exploit code	

Impact Gradient Labels Impact gradient labels describe the impact an exploited vulnerability might have on a target device, and they only apply to the vulnerability and target device. The labels of *Confidentiality*, C, *Integrity*, I and *Availability*, A, describe the functionality of the vulnerability and the security requirements of the target device. For each C, I and A, the labels include *None*, N, *Low*, L, and *High*, H, with the same ordered set $\{N, L, H\}$.

2.2.2 Adversarial Data

We primarily consider scenarios in which adversaries have access outside of a targeted network zone. Otherwise, if an adversary has internal access, they likely could use credentials rather than software vulnerabilities for attacks. However, we do model an insider threat in section 2.6, but we do so from a network zone on the fringe of the targeted system.

Adversary objects have capability labels assigned from Table 1, and these should be selected to match the real adversarial capability closely. For example, a threat actor on the Internet may have the capability to exploit vulnerabilities with *High* attack complexity, *Low* privilege, and *Unproven* exploitability. An insider threat may have *High* privileges, but only have exploit capability for *Low*

attack complexity and *High* exploitability.

2.2.3 Network Service and Network Reachability

Vulnerabilities that remote adversaries could exploit are usually associated with specific network services, e.g., a web service. The network service information is critical for associating vulnerability exposure with the firewall policy, which governs access to network services. Currently, the network service associated with a vulnerability is not released/reported in any standard format. Instead, security operators usually need to manually dig it out by reading vulnerability descriptions such as those released in the NVD. We will describe how to extract the network service information from vulnerability data in Section 2.3, and how to explore an adversary's network reachability to the target device and service in Section 2.4.

2.3 Network Service Extraction

The enabling factor for defining an adversary's ability to reach a vulnerability is the extraction of network service information from the vulnerability's features and descriptions. We first use machine learning to extract network services for vulnerabilities and then apply natural language processing (NLP) to boost results. This section describes the machine learning approach, the NLP approach, and how they are combined into one pipeline to identify network services.

2.3.1 Machine Learning-based Extraction

The machine learning portion of the pipeline uses a predictive decision-tree model over standard feature data from the NVD to predict the network services associated with vulnerabilities. Standard features include (i) Common Product Enumeration (CPE) [24], (ii) Common Vulnerability Scoring System (CVSS) [25] features, and (iii) the Common Weakness Enumeration (CWE) [27]. All of these features are regularly updated and made available by the NVD [64].

We initially tried using machine learning to predict for all network services. Machine learning by itself performs well for the *web* services and *client* services involving user interactions. How-

ever, it performs much worse for other network services, probably because there are relatively few vulnerabilities for other network services in the NVD dataset. Inspired by this observation, we use machine learning to classify vulnerabilities into three categories, *CLIENT*, *WEB*, and *INCONCLUSIVE*, for better accuracy. The *CLIENT* category represents the broad class of vulnerabilities in which either the adversary must exploit locally (e.g., local input) or must initiate client network traffic for a remote exploit (e.g., browser-based vulnerabilities). The *WEB* category represents vulnerabilities with web services. The *INCONCLUSIVE* category represents all other vulnerabilities in which machine learning does not accurately determine network services and which requires further processing by NLP.

We labeled network services for 19,433 vulnerabilities sampled from the 2017-2019 NVD dataset as our training data. The samples were shuffled and randomly partitioned into an 80% to 20% training-testing split. The model uses a decision-tree classifier using the features from the CVSS, CPE, and CWE described above and a Gini-index for branching. Table 2 shows the precision, recall, F-score, and support metrics for the machine learning model performance.

To explain these metrics, we consider the number of true positives (tp), true negatives (tn), false positive (fp), and false negatives (fn) with respect to the test dataset predictions matching the correct output. Precision is then defined to measure how well the model successfully predicts the output. This metric helps determine whether the false positive classification is acceptable.

$$\text{Precision} = \frac{tp}{tp + fp} \tag{1}$$

Recall measures the percentage of predictions in the test dataset successfully classified. This metric is useful when analysts have a significant concern for a model failing to classify correctly.

$$\text{Recall} = \frac{tp}{tp + fn} \tag{2}$$

Likewise, the F-score is the harmonic mean of the precision and recall scores to provide a more

holistic performance of the model.

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Table 2: Machine Learning Classification Results for Network Services

Network Service Type	Precision	Recall	F-Score	Support
CLIENT	100%	100%	100%	3,764
WEB	99%	100%	100%	591
INCONCLUSIVE	99%	99%	99%	504

2.3.2 Natural Language Processing-based Extraction

We then use NLP to further process the vulnerabilities within the *INCONCLUSIVE* category of the machine learning prediction. One approach is to directly classify each vulnerability description with a label identifying the network service. However, there are thousands of network services which makes it very challenging to get a high accuracy based on the currently available data. Instead, we build semantic meaning from the vulnerability descriptions in the NVD through named-entity recognition (NER), which locates and classifies named entities in a text into pre-defined categories such as organizations and products. For example, NER would classify "Google LLC" in a sentence as an Organization. For a complete description of NER, refer to [23, 79]. We use NER to extract standard features in vulnerability descriptions.

Inspired by existing work on cybersecurity ontologies [78, 81, 14], we define the following named entities for classifying network services:

1. **SERVICE** - Service affected by the vulnerability. Examples include HTTP, VNC, ssh, and CLI. These entities often map directly to network services.

2. **SOFTWARE** - Software product affected by the vulnerability. Software products often have network service requirements. For example, a vulnerability affecting WordPress maps to a web service and Google Chrome vulnerabilities require client interactions.

3. **THREAT** - Method used to exploit the vulnerability. This entity is most helpful in identifying web services. Descriptions of attack vectors commonly use HTTP and HTML terms such as POST, URI, and cookie. These adjectives often follow the preposition "via" in the description.

4. **WEAKNESS** - Software failure causing the vulnerability. Examples include web attack references such as CSRF, SSRF, and path traversal. These weaknesses commonly precede the term *vulnerability* as an adjective.

We annotated approximately 4,000 vulnerability descriptions from the 2017 through 2019 NVD dataset. Then a convolutional neural network (CNN) model for recognizing named entities was trained based on these vulnerabilities with a random 80% to 20% training-testing split. Table 3 shows the results.

We then build a set of rules for mapping vulnerabilities to network services using named entities. Each rule tags a specific network service based on the named entities extracted from vulnerability descriptions.

Table 3: NLP Named-Entity Recognition Scores

NLP	**NER Results Summary**		
NER Category	***Precision***	***Recall***	***F1-score***
SERVICE	76%	68%	72%
SOFTWARE	63%	64%	63%
THREAT	72%	61%	66%
WEAKNESS	70%	54%	61%

2.3.3 The Service Extraction Pipeline

The entire pipeline of network service extraction is as follows. We first use the above machine learning method to identify a set of vulnerabilities associated with the *WEB* and *CLIENT* services. For other vulnerabilities that fall into the *INCONCLUSIVE* category, the above NLP-based rule

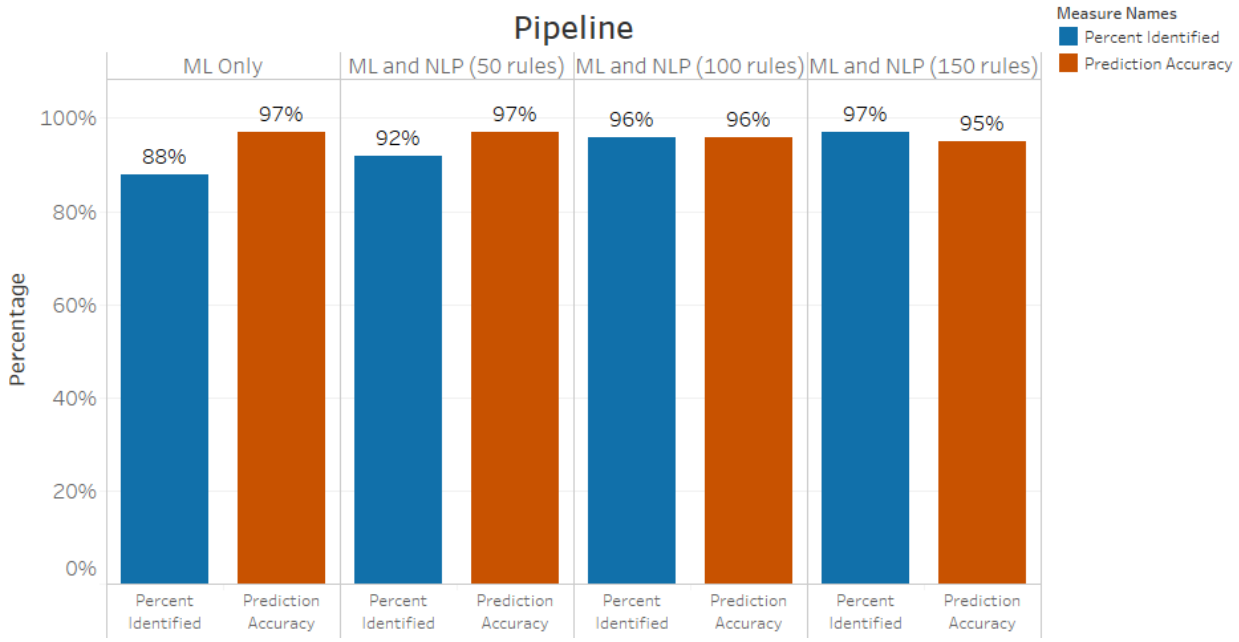

Figure 1: Performance of Network Service Extraction

matching identifies the specific network services. Vulnerabilities that do not match any NLP-based rules are left for manual analysis by security operators.

We tested the pipeline over 3,841 vulnerabilities published in the NVD in 2020. We used the 2020 NVD dataset for testing because both the machine-learning and NLP models trained over data features from 2017 through 2019. The results are shown in Figure 1.

The left-most column shows the results of *machine-learning only* where service classification is derived for 88% of vulnerabilities with a 97% classification accuracy (for the remaining 12% of vulnerabilities, the machine learning method alone is not able to generate any service classification). The following three columns show the classification results of *machine learning and NLP* when the number of NLP rules changes from 50 to 100 and 150. When there are 50 NLP rules, more vulnerabilities' network services are classified than machine learning only while the overall classification accuracy maintains at the same level. By adding NLP rules from 50 to 150, vulnerabilities with identified network services increase from 92% to 97%. As a trade-off, there is a slight reduction in the overall classification accuracy (from 97% to 95%) since some NLP rules generate wrong service mappings. However, the accuracy is still high.

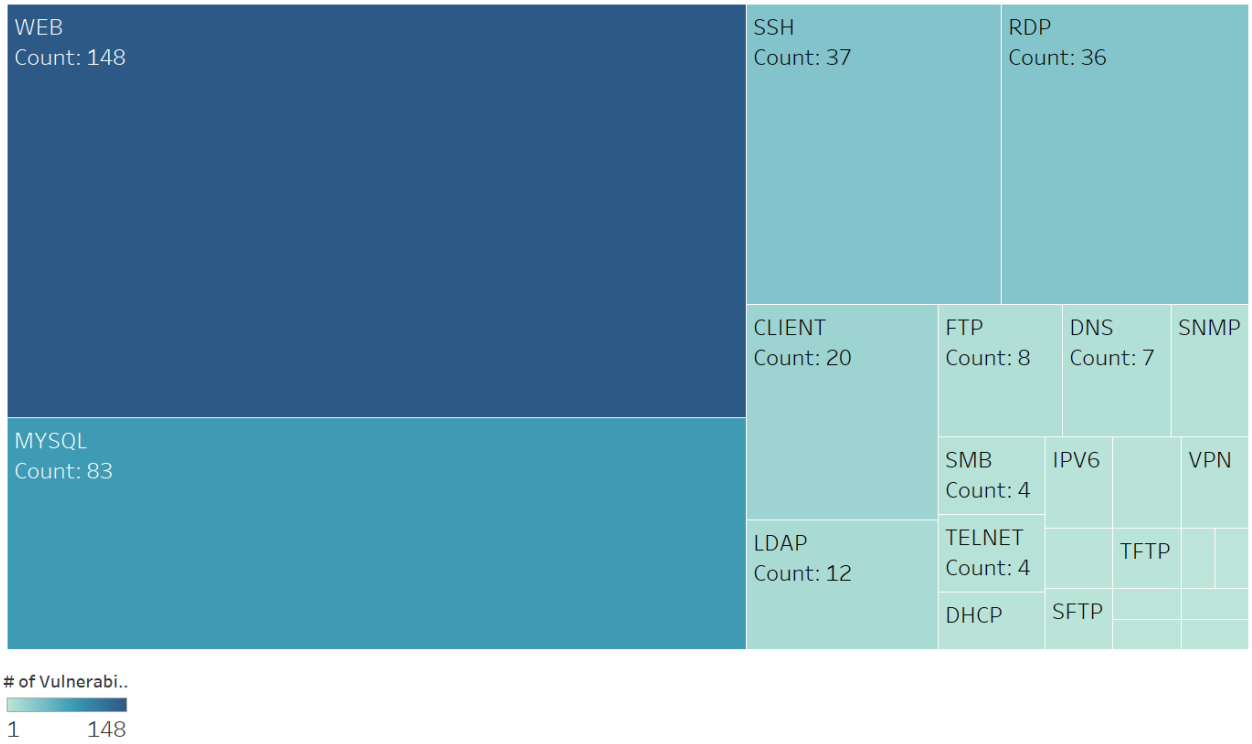

Figure 2: Additional Network Services Identified through NLP

3,529 (92%) of the identified network services were categorized as either *WEB* or *CLIENT*, with 3,353 (87%) identified by machine learning and 404 (10%) by NLP. Figure 2 shows the diversity of network services identified solely by NLP. The treemap shows the number of network services in both color and area.

Each network service maps to a set of transport-layer network ports. The ports directly associate with firewall rules to automatically assess network reachability, as we show in the next section.

2.4 Network Reachability

Network reachability means an adversary's ability to access a target device over a network. Firewalls between the adversary and target device serve as the principal inhibitor of access for most server environments. Determining the combined and effective access permitted by the set of firewalls is tantamount to establishing whether an adversary can reach a given vulnerability. Reach

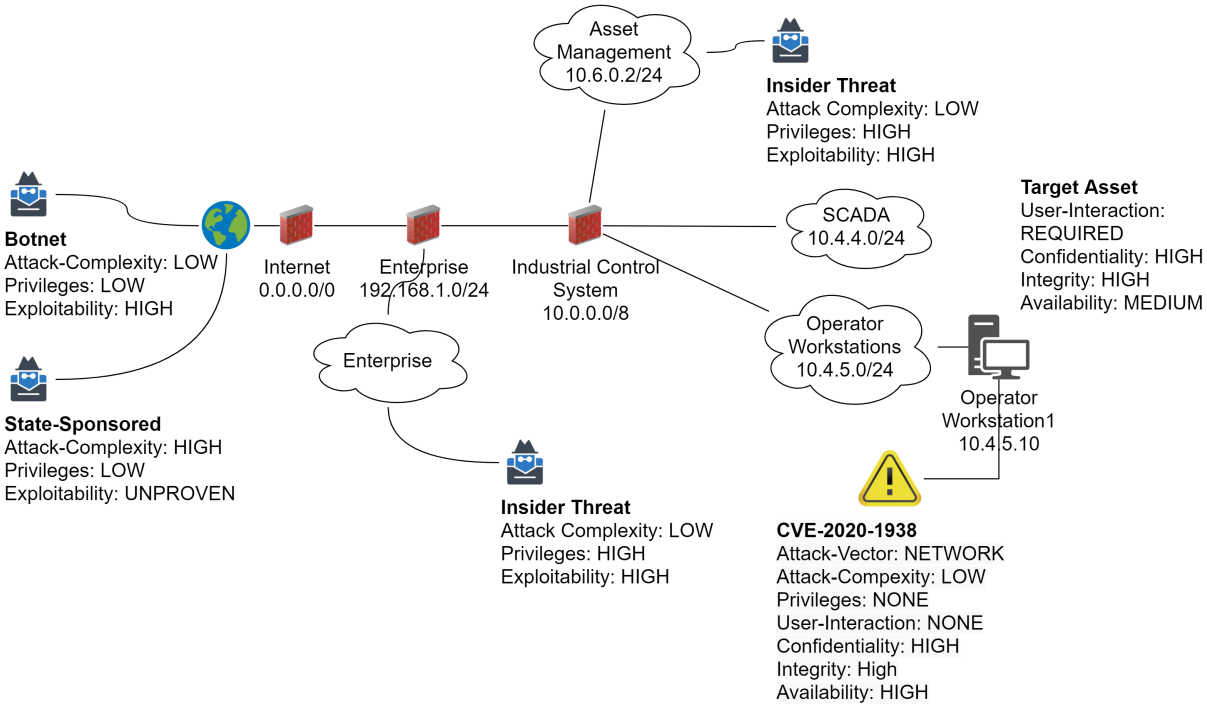

Figure 3: Reachability analysis combining Adversaries, Targets, and Vulnerabilities

analysis includes an assessment of both i) direct network service access and ii) interactive access, in which an adversary extends its reach into the network through the possession of authentication credentials and vulnerability exploits.

This step aims to identify combinations of adversaries, target devices, and vulnerabilities for safety analysis. As shown in figure 3, target devices reside in network zones, and adversaries get placed in network zones based on some realistic approximation of where an adversary may already reside in the network. In this diagram, the outmost firewall may block the state-sponsored adversary from the Internet to the target operator's workstation. However, the internal firewall policies may allow an insider threat to reach the target operator workstation.

The reachability model answers the question, "can an adversary reach a target asset and exploit a vulnerability?" Armed with each vulnerability's network service information, the model can use a firewall configuration analyzer (e.g., NP-View[1]) to parse out network topology and accessibility

[1]https://www.network-perception.com/

between network zones and determine whether a given adversary has an opportunity to exploit a given vulnerability.

Firewall configuration in the network can be parsed to produce paths represented as a five-tuple variable of protocol, source and destination IP address, and source and destination transport-layer port. The set of path tuples serve as an effective firewall ruleset between all network zones. We also further categorize firewall rules related to interactive services (e.g., SSH, RDP). This subset of rules allows the adversary to have authenticated access in a network zone, thereby extending its reach and pivot toward its target. In contrast, non-interactive services (e.g., HTTP, SMB) do not provide a direct opportunity for pivoting into a network zone.

We can now model adversarial reach by placing adversaries in network zones such as the placement shown in Figure 3. For assessing an adversary's ability to pivot between networks, the model uses an undirected graph because interactive access can occur between any network zones in a routed network. The network services used to permit interactive access, $\Psi \in \Gamma$, include those which permit the adversary to have local interactive access to the target operating system.

A depth-first-search with cycle detection traverses the graph to associate adversaries with network zones. Suppose an adversary can interact with a device in a different network zone because of permitted interactive access. In that case, the model assumes the adversary can obtain credentials in the existing zone. By recursively traversing the graph, adversaries copy over into each network zone to which it may pivot.

2.5 Model Checking Vulnerability Safety

Network reachability represents a significant obstacle for the adversary, but adversarial access to the vulnerability is not the end of the story. This section presents a definition of safety with respect to a vulnerability, adversary, and its target device. Throughout the model discussion, we refer to the transition system in the following definition:

Definition 2.1. *Software Vulnerability Transition System*

– S - Set of states

– $R \subseteq S \times S$ - Transition functions

– $S_0 \subseteq S$ - The set of initial states

– AP - Set of atomic propositions

– $L : S \rightarrow 2^{AP}$ - Labeling of states formula

– $\Phi = AG(\neg unsafe)$ - Invariant condition defining an secure state

– S_s - Set of final accepting states

The safety invariant, Φ, indicates the model cannot reach an unsafe state, which means the adversary cannot exploit the vulnerability. The labels apply to the three model objects: (i) vulnerabilities denoted as v, (ii) target devices denoted as τ, and (iii) adversaries denoted as ϵ. These objects are the basic building blocks for assessing system safety. Together, these objects provide the propositional labels applying to a system state, such that $AP = \{v, \tau, \epsilon\}$.

Figure 4 provides an example of how object labels combine in a final state, S_s, to calculate both the safety invariant and the overall impact. In this example, the vulnerability dominates the attack complexity (AC) of the adversary. A high AC for a vulnerability means an adversary with low AC could not successfully exploit the vulnerability. Therefore, as we show later in this section, the final state is safe. Because the final state is considered safe, the model does not assess impact. However, if the final state was unsafe, the calculated impact gradient label (see section 2.2.1) of *medium* would apply.

2.5.1 Dominance Relation in Capability State Labels

We now formally describe each capability label and how the labeling function applies to the vulnerability in the final state.

Definition 2.2 (Dominance Labels)**.** A vulnerability state label grouping in which the following properties hold:

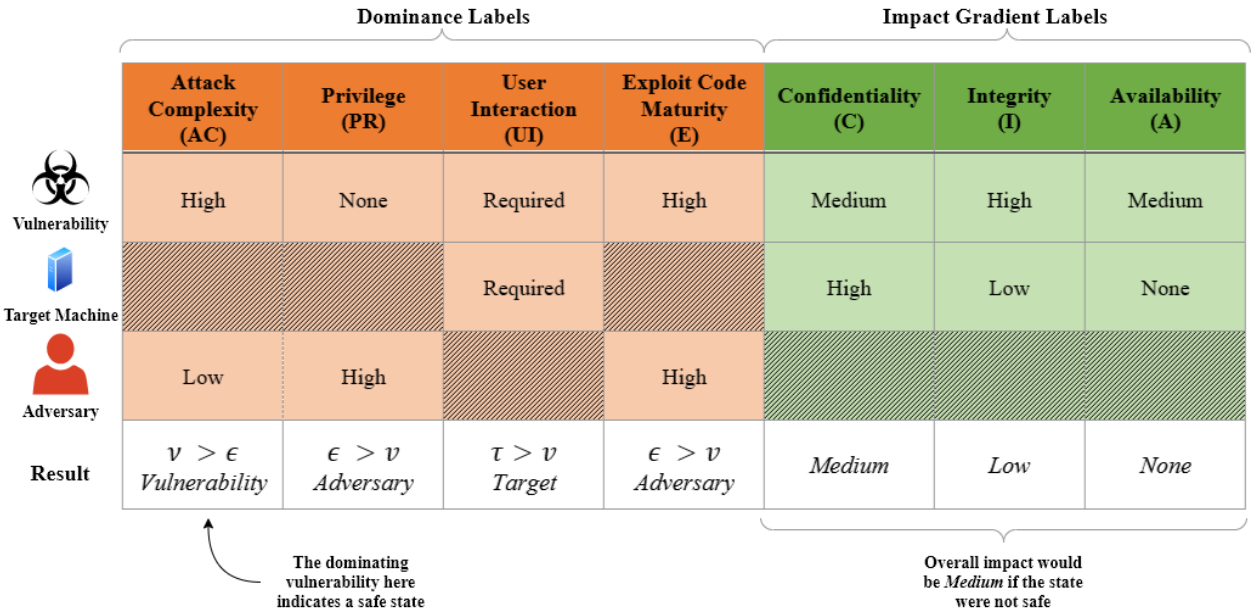

Figure 4: Example Final State Labeling

- Distinct labels in the group can order in terms of increasing difficulty and decreasing opportunity of exploitation
- A cumulative set hierarchy represents attacker capability on the ordered labels.
- The label group defines a necessary condition for exploitation.

This definition holds for the CVSS exploitability and temporal metrics. Labels have order applied as described below in this section. The cumulative set hierarchy follows from the ordered set, in which the capabilities accrue based on the order. Then, finally, the necessity for exploitation should be evident in the description of each label group.

We can now formally define safety in terms of the dominance relation between v, τ and ϵ. For a given state $s \in S$, $L(s)$ includes labels for $\{v, \tau, \epsilon\}$ in the capability categories of each $Cap = \{AC, PR, UI, EX\}$. We symbolize a state label for some category $c \in Cap$ with respect to an object as v_s^c, τ_s^c and ϵ_s^c. For brevity, Cap is also split as Cap_ϵ for labels applying to adversaries and Cap_τ for labels applying to targets. The dominance relation is defined for vulnerability dominance

as:

$$(v_s)dom(\epsilon_s, \tau_s) \iff$$
$$\exists c \in Cap_\epsilon, v_s^c > \epsilon_s^c$$
$$\vee \exists c \in Cap_\tau, v_s^c > \tau_s^c$$

And dominance for the adversary and target is defined as:

$$(\epsilon_s, \tau_s)dom(v_s) \iff$$
$$\forall c \in Cap_\epsilon, \epsilon_s^c > v_s^c$$
$$\wedge \forall c \in Cap_\tau, \tau_s^c > v_s^c$$

The safety invariant, Φ, is defined as the vulnerability dominating the target and adversary, meaning the adversary cannot exploit the target using the vulnerability. Likewise, a *safe* state means the adversary lacks some capability to exploit the vulnerability on the target device. The following theorem associates the dominance property to our definition of model safety.

Theorem 2.1. if $(v_s)dom(\epsilon_s, \tau_s)$, then the state, $s \in S_s$ is safe.

Proof. We begin proving this by assuming the dominance relation holds between vulnerabilities and adversaries. Each capability-based category forms an ordered set which is also a cumulative set hierarchy with respect to v, ϵ and τ.

$$\forall c \in Cap \mid 0 \leq i < |c|,$$
$$v^{c_i} \subseteq v^{c_{i+1}}, \epsilon^{c_{i+1}} \subseteq \epsilon^{c_i}, \tau^{c_{i+1}} \subseteq \tau^{c_i}$$

Recall that the set order indicates both (i) increasing difficulty and (ii) decreasing exploit opportunity. For v, lower ordered categories are a subset of those higher-ordered. The ordering means an unsafe vulnerability based on v^{c_i} remains unsafe for any lower ordered capability. In contrast,

ϵ and τ have a reverse hierarchical set because capability at a higher level would suffice to exploit any vulnerability with v at a lower order.

Because v is dominating, we know there is at least one $c \in Cap$ in which v is greater than either ϵ or τ. Through the cumulative set hierarchy, it follows that:

$$\exists c \in Cap \mid v^c \cap \epsilon^c \in \emptyset \vee v^c \cap \tau^c \in \emptyset \tag{4}$$

Therefore, the adversary cannot exploit the vulnerability on the target in at least one category, and by definition 2.2, v is safe. The proof ends. □

The converse is not necessarily true because some other capability category may exist outside of the CVSS metric.

2.5.2 Measuring Impact

Impact gradient labels apply when the final state of a system is not safe. These labels provide further context for assessing the vulnerable state of a system and prioritizing risk mitigation work. The set of risk gradient categories are $G = \{C, I, A\}$. Similar to dominance labels, we also define each label category as a cumulative set hierarchy in which:

$$\forall g \in G \mid 0 \leq i < |g|, g_i \subseteq g_{i+1} \tag{5}$$

For example, if the vulnerability label for confidentiality were H (or high), then $v^C = \{L, M, H\}$. Now, a simple impact gradient calculation provides the combined result of v and τ:

Definition 2.3. A calculated impact gradient label applies to final states S_s in which $\Phi = AG(\neg safe)$ as:

$$Impact(S_s) = max(\bigcup_{g \in G} v^g \cap \tau^g)$$

The impact calculation bounds the impact of the target device label.

2.6 Evaluations

Open data sets for firewall and vulnerability management are not available due to the highly sensitive nature of the data and industry-specific compliance obligations. To overcome these barriers, we generate a realistic sample system. We adopt an approach to generate the requisite system data using network service exploration. The applications required to run on the system determine the required network services. We identified the required applications through interviews, assessments, and exploration of industry compliance obligations. These applications are decomposed into classes of commonly used computing assets and further decomposed into individual assets and software.

In particular, our sample system derives from applications and compliance obligations required for a power grid control center, but the approach works for other critical infrastructure domains as well. We derive it using elements of the computing environment required by the North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) regulatory standards [68]. The standard sufficiently references specific types of technology related to security and reliability for creating a representative sample. The resulting system contains 124 devices organized into 25 asset groups, e.g., Web Servers and Operator Workstations, as shown in Figure 5. The data set also includes 4,894 combined software assets mapped to the NVD.

We tested the implementation of network reachability and safety analysis on the above asset dataset using the vulnerabilities from the NVD for January 2017 through July 2020 that match the assets. Vulnerabilities map to software and computer assets using a combination of Common Product Enumeration (CPE) applicability matching, Microsoft vulnerability reports, and Red Hat vulnerability reports. In that timeframe, we found a total of 106,313 vulnerabilities applicable to the assets.

To generate firewall rules, we analyze each asset by identifying its listening network services, client services, and remote access services and then generate firewall rules for these services. The generated firewall ruleset has a realistic basis in the system's common sector-specific services. We generated 1,156 distinct firewall rules by traversing the network graph and filling in the required

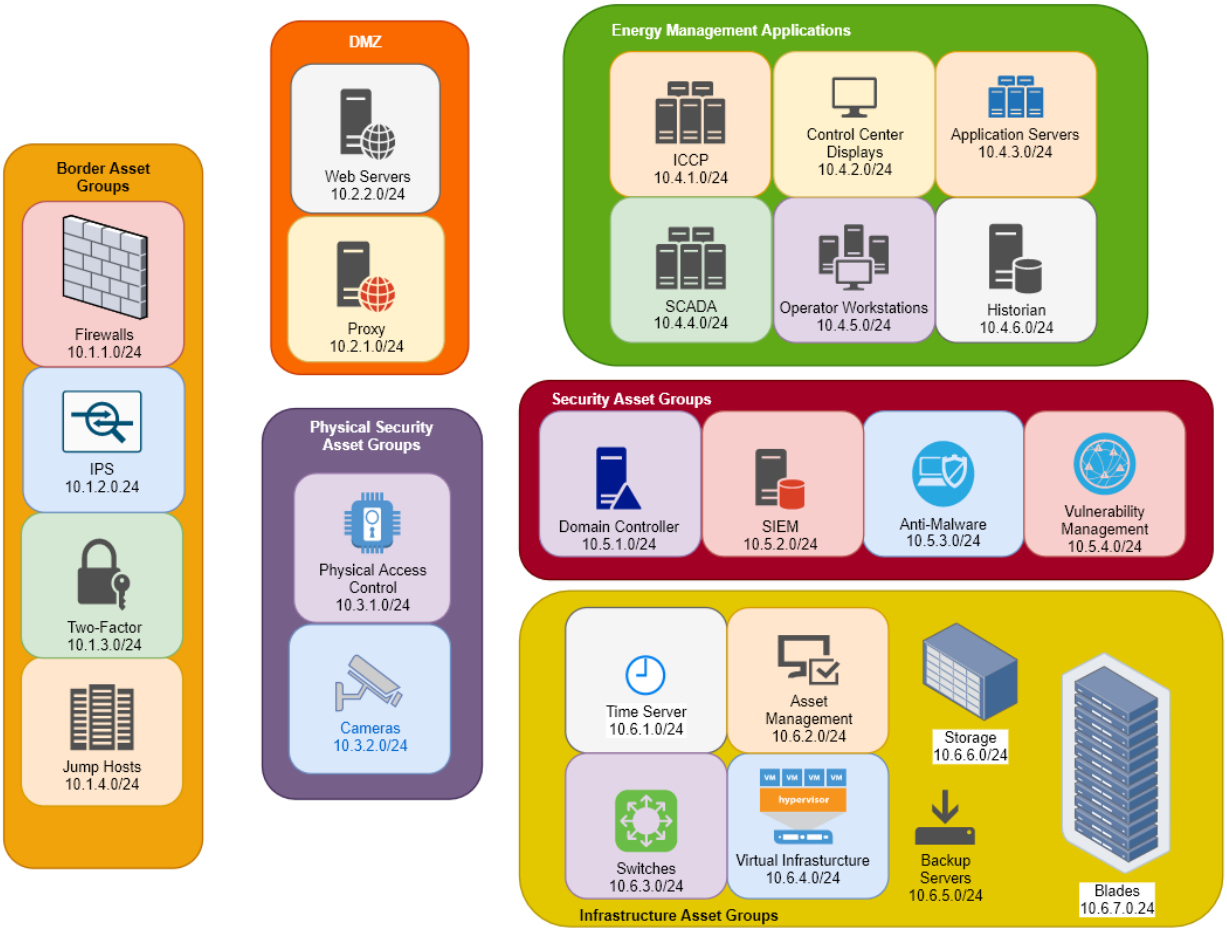

Figure 5: Sample System Network Zonal Diagram

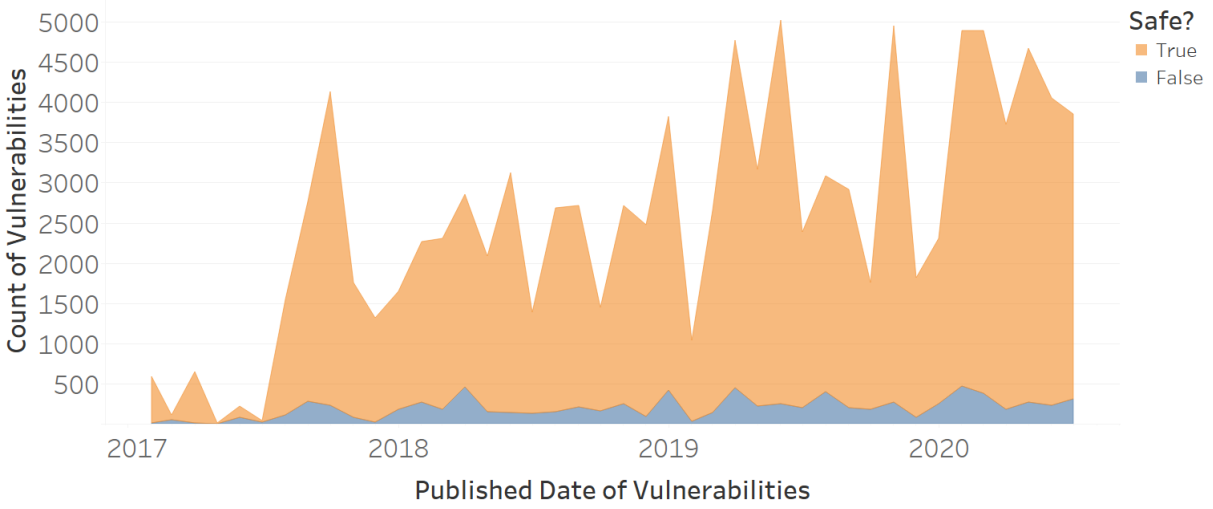

Figure 6: Monthly Safety Analysis of all Applicable Vulnerabilities

services.

Adversaries had access to the Internet network zone, enterprise network zone, and an asset management zone internal to the control system in the model. The Internet adversaries modeled a state-sponsored adversary (i.e., skillful with minimum internal privileges) and an automated botnet (i.e., minimally capable with minimum internal privileges). The two internal adversaries modeled inside threats that hold highly privileged access but are minimally capable of exploiting vulnerabilities.

Each network zone's data structure included all adversaries having interactive reach into the zone based on the generated firewall ruleset. Network services were extracted for all the vulnerabilities as well.

Finally, each of the 106,313 vulnerabilities received an assessment using the presented model-checking safety analysis. The assessment included modeling each adversary with interactive access to the device and assessing adversaries having reach associated with the vulnerability's network service. A particular case also occurs for vulnerabilities requiring user interactions. Our model mainly considers inbound reachability from the Internet to a target server device. However, we model outbound user interaction by assuming the worst-case scenario, in which a state-sponsored adversary has backdoor interactive access to an asset.

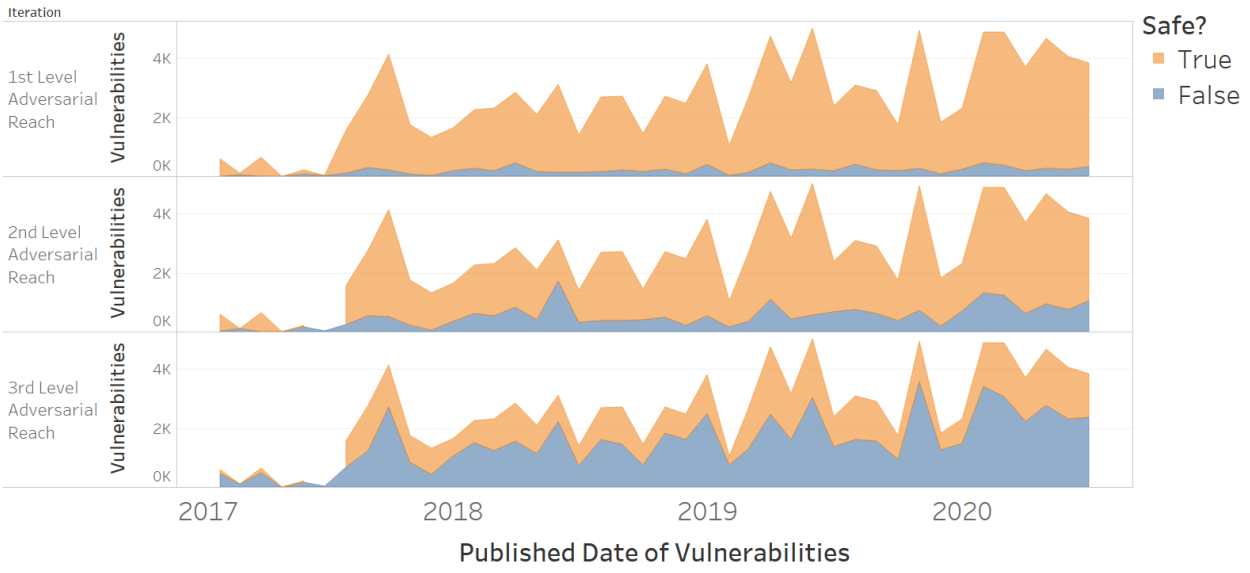

Figure 7: Iterative Safety Analysis for all Applicable Vulnerabilities

Figure 6 presents the results. This graph shows the monthly count of both safe and unsafe vulnerabilities for the sampling period. Those vulnerabilities assessed as safe account for approximately 92% of the vulnerabilities overall, whereas those assessed as unsafe remain consistently below 500 applicable vulnerabilities per month. *This implies security operators informed by safety analysis can use their limited resources to address unsafe vulnerabilities.*

The model checking also allows iterative exploration where the adversary's reach extends through unsafe vulnerabilities, which we term gateway vulnerabilities. Gateway vulnerabilities allow full access or privilege escalation on a reachable target such that exploitation would extend the adversary's reach into additional network zones. The model checking increases reachability only for unsafe vulnerabilities having *High* integrity impact. In contrast, vulnerabilities with only denial of service effects (i.e., availability impact) or information disclosure effects (i.e., confidentiality impact) do not extend adversary reach.

Figure 7 shows the results of extending reachability using gateway vulnerabilities. The first graph/iteration is a copy of Figure 6, and the second graph/iteration shows the number of increased vulnerabilities after extending adversarial reach from the first iteration. The third graph is the third iteration. It is the full extension of adversarial reach since there are no additional gateway

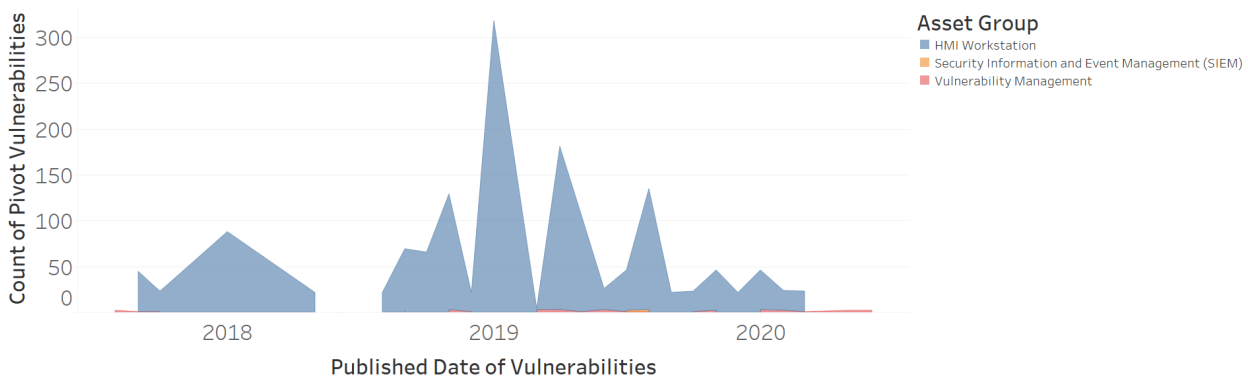

Figure 8: Vulnerabilities Allowing Extension of Adversarial Reach

vulnerabilities beyond the third iteration. The number of unsafe vulnerabilities rises from 8% in the first iteration to 20% in the second, and then 60% in the final iteration and maximum adversarial reach.

The data would suggest an adversarial advantage in unsafe vulnerabilities, but a countermeasure strategy to immediately mitigate gateway vulnerabilities would maintain the defense advantage of minimal unsafe vulnerabilities. The graph in Fig. 8 shows the number of gateway vulnerabilities per month, which remains minuscule compared to the overall number of vulnerabilities.

2.6.1 Discussion of Results and Limitations

Our results show the substantial number of vulnerabilities applying to a typical system from month to month. We have defined a network attack model based on realistic assumption for routing and firewall placement and proposed a definition of safety as a function of both adversarial capability and reachability. The simulation results demonstrate the considerably small number of vulnerabilities which may lead to an unsafe state. These results suggest an effective defensive strategy of tightly coupling firewall countermeasures with vulnerability mitigation to focus resources on eliminating unsafe vulnerabilities. Furthermore, focusing resources to eliminate gateway vulnerabilities has an even greater effect to limiting adversarial reach.

The foremost limitation of this model is the basis for selecting adversarial capabilities. Vulnerability safety changes significantly with broad adversarial network reachability. For example,

if a nation state adversary has inside access to critical network zones, then many more vulnerabilities would naturally create an unsafe system state. However, one might also assume an adversary deeply embedded in the system has little need for exploiting vulnerabilities to accomplish their objective.

Further limitations may exist in the data we use to model adversarial capability in the CVSS. The use of attack vectors, complexity, privileges, and exploitability appears well founded and accurate, but we could not find a public data set to validate a claim these closely model adversarial capability.

2.7 Conclusion

We proposed a new scheme of automatically evaluating software vulnerabilities for state safety using firewall configuration data. It involves automated identification of network services associated with vulnerabilities and connects that to firewall rules, target devices, and adversarial attributes under one formal framework. This model expands on existing network attack models by assessing specific vulnerabilities and target devices combined with adversarial attributes. With the NVD CVSS data as a basis, we expand to provide contextual information of a given system and threat scenario. In so doing, each applicable vulnerability can be assessed more tightly for overall system safety. The model tested on a simulated power grid control system using a methodology to generate specific assets representative of the given sector's services and compliance obligations. Tests on a realistically simulated sample system showed only 8% of the applicable vulnerabilities are unsafe. We further modeled the dynamic threat movement to pivot deeper into the network using gateway vulnerabilities and found the presence of gateway vulnerabilities, when left unmitigated, remarkably changes the number of unsafe vulnerabilities. The results suggest new strategies in cybersecurity operations to better apply limited resources

References

[4] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. "Scalable, Graph-Based Network Vulnerability Analysis". In: *ACM Conference on Computer and Communications Security*. 2002, pp. 217–224.

[8] M. Audinot, S. Pinchinat, and B. Kordy. “Guided Design of Attack Trees: A System-Based Approach”. In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. July 2018, pp. 61–75.

[14] Rich Piazza Bret Jordan and Trey Darley, eds. *STIX™ Version 2.1*. https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.html. Accessed: 2021-03-09. Mar. 2020.

[22] Keith Collins. “The hackers who broke into Equifax exploited a flaw in open-source server software”. In: *Quartz* (Sept. 8, 2017). URL: https://qz.com/1073221/the-hackers-who-broke-into-equifax-exploited-a-nine-year-old-security-flaw/ (visited on 09/14/2017).

[23] Ronan Collobert et al. “Natural language processing (almost) from scratch”. In: *Journal of machine learning research* 12.ARTICLE (2011), pp. 2493–2537.

[24] *Common Product Enumeration Standard*. https://nvd.nist.gov/products/cpe. Accessed: 2020-01-28.

[25] *Common Vulnerability Scoring System Specification*. https://www.first.org/cvss/v3.1/specification-document. Accessed: 2020-01-28.

[26] *Common Vulnerability Scoring System v3.1: Specification Document*. https://www.first.org/cvss/v3.1/specification-document. Accessed: 2020-02-01.

[27] *Common Weakness Enumeration*. https://cwe.mitre.org/. Accessed: 2020-01-28.

[36] B. Fila and W. Wideł. “Efficient Attack-Defense Tree Analysis using Pareto Attribute Domains”. In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. June 2019.

[38] M. Gamarra et al. “Analysis of Stepping Stone Attacks in Dynamic Vulnerability Graphs”. In: May 2018, pp. 1–7.

[40] Nirnay Ghosh and S. K. Ghosh. “An Approach for Security Assessment of Network Configurations Using Attack Graph”. In: *International Conference on Networks Communications*. 2010, pp. 283–288.

[47] Philip Huff et al. “A Recommender System for Tracking Vulnerabilities”. In: *International Workshop on Next Generation Security Operations Centers (NG-SOC)*. 2021.

[50] Barbara Kordy et al. “Foundations of attack–defense trees”. In: *International Workshop on Formal Aspects in Security and Trust*. 2010, pp. 80–95.

[53] Carl E Landwehr. “Formal models for computer security”. In: *ACM Computing Surveys (CSUR)* 13.3 (1981), pp. 247–278.

[55] H. T. Le and P. K. K. Loh. “Using Natural Language Tool to Assist VPRG Automated Extraction from Textual Vulnerability Description”. In: *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*. Mar. 2011.

[56] Triet Huynh Minh Le, Bushra Sabir, and Muhammad Ali Babar. "Automated software vulnerability assessment with concept drift". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019, pp. 371–382.

[58] H. Mantel and C. W. Probst. "On the Meaning and Purpose of Attack Trees". In: *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. June 2019, pp. 184–18415.

[59] Sjouke Mauw and Martijn Oostdijk. "Foundations of Attack Trees". In: *Information Security and Cryptology - ICISC 2005*. 2006, pp. 186–198.

[60] Kylie McClanahan and Qinghua Li. "Automatically Locating Mitigation Information for Security Vulnerabilities". In: *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids*. 2020.

[64] *National Vulnerability Database Data Feed*. https://nvd.nist.gov/vuln/data-feeds. Accessed: 2020-01-28.

[65] *National Vulnerability Database Data Feed*. https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time. Accessed: 2020-02-01.

[67] Steven Noel and Sushil Jajodia. "Metrics Suite for Network Attack Graph Analytics". In: *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. 2014, pp. 5–8.

[68] *North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) Standards*. https://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx. Accessed: 2020-02-01.

[70] Cynthia Phillips and Laura Painton Swiler. "A graph-based system for network-vulnerability analysis". In: *Proceedings of the 1998 workshop on New security paradigms*. 1998, pp. 71–79.

[76] Bruce Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[78] Leslie F Sikos. "OWL ontologies in cybersecurity: conceptual modeling of cyber-knowledge". In: *AI in Cybersecurity*. Springer, 2019, pp. 1–17.

[79] *spaCy Linguistic Features*. https://spacy.io/usage/linguistic-features. Accessed: 2021-03-15.

[81] Zareen Syed et al. "UCO: A unified cybersecurity ontology". In: *UMBC Student Collection* (2016).

[84] *Vulnerability and Patch Management Resources*. http://cybersecurity.ddns.uark.edu/vpm/. Accessed: 2021-06-25.

[86] Lingyu Wang et al. "k-zero day safety: Measuring the security risk of networks against unknown attacks". In: *European Symposium on Research in Computer Security*. 2010, pp. 573–587.

[87] Peichao Wang et al. “Intelligent Prediction of Vulnerability Severity Level Based on Text Mining and XGBboost”. In: *2019 Eleventh International Conference on Advanced Computational Intelligence (ICACI)*. 2019, pp. 72–77.

[88] Jeannette M Wing et al. “Scenario graphs applied to network security”. In: *Information assurance: survivability and security in networked systems* (2008), pp. 247–277.

[90] Anming Xie et al. “Applying Attack Graphs to Network Security Metric”. In: *Proceedings of the 2009 International Conference on Multimedia Information Networking and Security - Volume 01*. 2009, pp. 427–431.

[91] M. Xu et al. “Dominance as a New Trusted Computing Primitive for the Internet of Things”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019.

[94] Fengli Zhang and Qinghua Li. “Dynamic Risk-Aware Patch Scheduling”. In: *IEEE Conference on Communications and Network Security (CNS)*. 2020.

[96] Fengli Zhang et al. “A Machine Learning-based Approach for Automated Vulnerability Remediation Analysis”. In: *IEEE Conference on Communications and Network Security (CNS)*. 2020.

3 A Recommender System for Tracking Vulnerabilities

3.1 Introduction

An organization wishing to mitigate its software vulnerabilities has clear disadvantages against an adversary. No obvious tie exists between the installed software of the entity and the software publishers that maintain the vulnerability reporting and mitigation for a given piece of software. The installed software and hardware inventory observed by a consumer is referred to as *software inventory package name*. Organizations have tens of thousands of software packages represented by hundreds to thousands of software publishers. These software publishers vary significantly from some of the largest companies on earth to mostly abandoned software projects.

Most cybersecurity operation centers have a process to identify vulnerabilities through a combination of scanning, asset management, and assessments. However, the fastest and surest strategy in preventing vulnerabilities from being exploited is an immediate notification from the vendor. For some vendors, this may occur through a paid support contract. Prominent operating system vendors, such as Microsoft, Apple, and Android, regularly push new vulnerability patches, which consumers then automatically deploy. However, many more software publishers rely on public notification of vulnerabilities through services such as the National Vulnerability Database (NVD).

In these cases, consumers can only act on public notifications by matching the vulnerable product to their hardware on software inventory. Timely analysis and remediation are critical in computing environments such as data centers, industrial control systems, and Internet of Things (IoT) systems.

The matching process is more complicated than it sounds. Hardware and software inventories often represent an amalgamation of different software publishers packaged up and deployed together. Then, the public vulnerability notifications use a standard naming convention which is well-formed but distinct from software package listings in the system inventory.

To further complicate the problem, software and hardware device inventory is a private data set. Organizations cannot simply post their software inventory for assistance in matching vulner-

ability reports. Revealing the deployed hardware and software names provides reconnaissance information adversaries might use to breach an organization, and revealing the data often violates regulatory requirements. Instead, cybersecurity analysts must painstakingly match their deployed software to public vulnerability reports.

Automatically matching enterprise software inventory with vulnerability reports frees up time for cybersecurity operations, but it also allows the convergence of the vulnerability data features with the asset and operational data features. Bringing these together paves the way for even more automation in vulnerability management and mitigation [96, 60].

Our approach solves the matching problem through a pipeline of natural language processing (NLP), fuzzy matching, and machine learning. A cybersecurity analyst using our technique can immediately obtain a shortlist of candidate matches to their software inventory or conclude the absence of any matching vulnerability identification source. While not perfect, the automation mimics the work of a human analyst to obtain a situational understanding of their environment and dramatically shortens the time to make a match decision. *To the best of our knowledge, this is the first work to solve the aforementioned matching problem.*

This chapter is organized as follows. We present prior work in section 3.2 and then describe the vulnerability reporting process and software name matching problem in section 3.3 from the perspective of cybersecurity operations. We present our recommender solution in section 3.4. Finally, we present our implementation and test results in section 4.6 and then conclude this chapter.

3.2 Prior Work

[33] defines software vulnerabilities as specific flaws in a piece of software that allows attackers to do something malicious. The problem of determining whether the software is vulnerable is undecidable [39], and as [52, 95] observe, vulnerabilities have become a normal process of cybersecurity operations. Entities with the responsibility of protecting against software vulnerabilities face the challenge of timely identifying vulnerabilities through public vulnerability alert repositories.

Researchers have studied the problem of public vulnerability reporting in several contexts. [73] examines the time delay between the reporting of vulnerabilities and the reporting of Common Vulnerability Scoring Scheme (CVSS) attributes. Several works enhance publicly reported CVSS data using machine learning and NLP to obtain new features and improve vulnerability and patch management decisions. [96] uses the CVSS metrics to automatically recommend vulnerability remediation actions. [94] uses vulnerability features to predict the probability of exploit for vulnerabilities and optimize patch scheduling. [60] uses natural language processing to automatically localize vulnerability information from online resources. In [55], a model is provided for NLP Named Entity Recognition (NER) over vulnerability descriptions to extract cause, consequence analysis, and impact ratings automatically. [92] demonstrates temporal difference over time in the vulnerability CVSS when used for machine learning. In [87], machine learning improves the accuracy of impact scoring, and [56] shows the drift over time of NLP models when applied to vulnerability descriptions. Similarly, [74] parses the CVE description to obtain vulnerability characteristics automatically, and [46] extracts network services from vulnerability descriptions using NLP to automatically perform threat analysis.

However, entities cannot realize the benefits of enhanced vulnerability intelligence without mapping the Common Vulnerabilities and Exposures (CVE) to the software inventory of an entity through a Common Platform Enumerator (CPE). For this task, we look to recommender systems. The term *recommender system* originated with [71] as an approach used to filter through large datasets to present the most relevant and desired selection back to the user. [16] defines recommender systems as those guiding the user to interesting or useful objects in a large space of possible options. Although most recommender systems [12] focus on user experience, there has been recent work to apply the approach to cybersecurity [2, 37, 66]. To the best of our knowledge, this is the first approach to address the problem of matching raw software inventory to standardized software product names.

3.3 Background

Currently, the largest open vulnerability database is maintained as the U.S. National Vulnerability Database (NVD) with over 1,500 vulnerability being reported monthly. Each of these vulnerabilities contains a section of products to which the vulnerabilty applies. The NVD relies on CVE Numbering Authorities (CNAs) to report vulnerabilities for products for which the CNAs have responsibility. As of this writing there are 167 CNAs representing 28 countries. A summary of the top ten CNAs reporting the most vulnerabilities in 2020 is provided in Table 4.

Table 4: Estimated Number of Vulnerabilities Reported in 2020 by CNA

CVE Numbering Authority	**Reported**
MITRE Corporation	3,669
Microsoft Corporation	1,082
Oracle	807
Cisco Systems, Inc.	436
Android (Google Inc. or Open Handset Alliance)	413
GitHub, Inc.	407
IBM Corporation	353
Adobe Systems Incorporated	308
Apple Inc.	272
ICS-CERT	225
Jenkins Project	209

When reporting vulnerabilities, CNAs identify the new vulnerability by a Common Vulnerability Enumerator (CVE). The CVE list is an open database sponsored by the United States Department of Homeland Security [2]. The CVE is commonly used by vulnerability databases, bug bounty programs, and exploit databases. The NVD uses the Common Platform Enumeration (CPE) software product naming standard when reporting vulnerability product applicability.

The CPE is based off the Uniform Resource Identifier (URI) syntax and provides a set theoretic naming convention in which a source and target software product name may be structured and compared as sets [21]. The naming standard is most commonly used to define the set of software to which a vulnerability applies using the following keys:

[2]https://cve.mitre.org/cve/

- Part - Type of software. A value *a* indicates software, *h* indicates hardware, and *o* indicates operating system.
- Vendor - Name of the manufacturer or publisher of the software.
- Product - Name of the product.
- Version - Version of the product.
- Update - A major update of the product such as a service pack for Windows. Currently, this key is only specified in 10% of the products reported to the NVD.
- Edition - Used to specify different types of the same product. However, in practice, the edition is commonly specified in the product name.

The CPE URI reads from general to specific in the format cpe:2.3: <part >: <vendor >: <product >: <version >: <update >: <edition > using the key value indicated in the URI. A '*' character is used as a wildcard for the source CPE. When matching sources to target URIs, the wildcard defines a superset based on other key value pairs in the URI. For example *CPE:2.3:a:microsoft:*:*:*:** defines the superset of all Microsoft software.

Vulnerabilities in the NVD define software applicability using CPEs in a boolean expression. In most cases, this means a simple *OR* operation to define applicability to multiple products. However, certain CNAs use the *AND* operation when the vulnerability only applies to a certain platform of the product. For example, a vulnerability for Adobe Acrobat on Android as opposed to the Windows operating system, would use the boolean *AND* operation to distinguish the applicability. An example with AND and OR operations is the Wordpress vulnerability CVE-2020-10257 [3] that has 62 vulnerable configurations.

The intention of the NVD is for organizations to match their hardware and software inventory to CVEs and receive the applicable vulnerability notifications. However, in most cases, there is no

[3]https://nvd.nist.gov/vuln/detail/CVE-2020-10257

clean way to do so. Table 5 provides a summary of CNAs who provide an online mapping between CVEs and software products and packages.

Table 5: CVE Notifications in Practice

Hyperlinked CNA	**Matching Approach**
Cisco, Inc.	Each Cisco advisory is mapped to multiple CVEs, but product matching is not known to be automated
Debian	CVEs are listed by package name, which can be matched to Debian APT listing
Intel Corp.	Each Intel advisory is mapped to multiple CVEs, but product matching is not known to be automated
Microsoft Corp.	A spreadsheet can be downloaded from the source URI with Microsoft Knowledge Base (KB) mapping to CVEs. Product matching comes through their Windows Security Update Service (WSUS) tool which can be configured to output a KB listing.
RedHat, Inc.	CVEs are mapped to packages, and product listing can be matched from the output of RedHat Package Manager.
Suse	Package to CVE mappings can be scraped from the given URI, and packages are listed on the operating system through RedHat Package Manager.
Ubuntu	Package to CVE mappings can be scraped from the given URI, and packages can be matched through an APT listing on the operating system.

As shown, most Linux operating systems can automatically match to CVEs using package listings available through the operating system. Cisco, Intel, and Microsoft provide up to date mappings from update packages to the CVE. Microsoft is unique in providing a tool for automatically extracting applicable packages (i.e., KBs) for an operating system.

As far as we can tell, no other CNAs provide an automated software inventory to CVE listing. Therefore, organizations wanting to receive notifications of new vulnerabilities are left to manually map their remaining inventory to CPE source URIs. Alternatively, they may review all new vulnerabilities for applicability, but the mapping still must regularly occur to perform vulnerability patching and mitigation.

3.4 Fuzzy Matching Technique

This section presents a solution for automatically matching target CPEs against an enterprise hardware and software inventory. Doing so not only allows for automated vulnerability notifications but also enables organizations to combine data features between vulnerabilities and vulnerable assets to improve cybersecurity risk understanding and mitigation decisions.

The approach to fuzzy matching approximates the process a security analyst would take in finding matching CPEs for their hardware and software inventory. A security analyst seeking to map their unknown software package inventory to a vulnerability source begins first by finding a suitable vendor name and filtering down further to find the product, but the search becomes messier when they cannot easily find a suitable source CPE for a specific product. If a vendor match is found, then the target CPE set might contain only the vendor, which is too coarse-grained. For large vendors, this might produce several hundred false positive CVE notifications each year.

We account for the complexity in decision making by determining candidate target CPEs through a three phase automation process. First, we extract the word vectors using the core english vocabulary from spaCy[4]. Then, based on the word vectors, we fuzzy match the software inventory package names to a set of CPEs using multiple metrics. Since many results might be returned in this step, finally, we use machine learning to order these CPEs and produce a small set (e.g., 5 to 10) of candidate target CPEs for a human to select.

3.4.1 Natural Language Processing

The inherent variation in natural language means that computers struggle to extract meaning in language. Irregular verb conjugations, for example, are difficult to detect using word stems. In our work, we find that software names, in particular, are often not dictionary words, and so many pre-trained or ready-to-use models have no context with which to recognize them.

Because of this, there is great value in representing words in some standardized, machine-readable format; Word2Vec is a recent but well-established method for this task [62]. Word2Vec

[4]https://spacy.io/

is an algorithm which processes a text corpus and projects the corpus vocabulary as vectors in a multi-dimensional space. Word2Vec can be trained on any corpus and will create word vectors for all tokens within it. Using word vectors, many NLP tasks become programmatic in nature. Measures of vector similarity can predict synonyms or antonyms. Similarly, word vectors can be used to infer analogies. For example, if a direction is calculated between the vector for "man" and the vector for "king", that direction can be applied to the vector for "woman" to obtain "queen".

Word vectors are simply an array of the same size as the number of dimensions, often normalized in the range $[-1, 1]$ before storage or comparison. Once a Word2Vec model has been trained, only the word vectors must be stored, minimizing the storage space needed.

3.4.2 Fuzzy Matching

String similarity tests occur through multiple calculations. First, the similarity between the software package inventory and the CPE is measured using the cosine between two-word vectors A and B as shown in the following equation where equal vectors have a cosine angle of one, and maximally dissimilar vectors have a cosine of 0.

$$\cos\Theta = \frac{A \cdot B}{\|A\|\|B\|} \tag{6}$$

Software names commonly have multiple words and symbols. The comparison is normalized by first splitting the string by common stop words and then removing unnecessary symbols. Subsequently, calculating the cosine similarity occurs through the average of all distinct word vectors in the string. This approach allows for similar words to appear in a different order between two strings without impacting the similarity score.

However, the concatenation of vendor and product strings can have misleading cosine similarity scores. For instance, software products commonly get bought out by other companies, but the CPE name may remain the same, or product names can span several words and adversely impact the matching vendor. Consequently, the cosine similarity between the software package inventory

name A, and the vendor B_{vendor}, and the cosine similarity between the software package inventory name A and the product $B_{product}$ are calculated separately.

Finally, software package inventory names may not have a word vector representation due to misspellings or other variance in the software naming. Likewise, the cosine similarity calculation returns 0 even when the words closely align. In these cases, the similarity gets measured as the cosine similarity of the strings using the letter count where c represents the letter, and $C(x)$ represents the count of the letter x.

$$\cos\Theta = \frac{\sum_{\forall c \in A \cap B} C(A.c) \times C(B.c)}{\sqrt{C(A.c)^2} \times \sqrt{C(B.c)^2}} \tag{7}$$

3.4.3 Machine Learning

Our matching solution uses machine learning as the final step in the pipeline to better distinguish the results of fuzzy matching. The machine learning model assists in ordering the results from fuzzy matching and presenting a smaller focused set to human operators. This approach improves the accuracy of the identified vulnerabilities by considering both fuzzy matching and historical vulnerability reporting for the software product.

The number of vulnerabilities between different vendors and different products varies greatly. For example, accidentally matching the vendor Goomeo instead of Google for a software package drastically affects the number of accurately identified vulnerabilities because Google has several vulnerabilities identified each month. Thus, using the string similarity in the previous fuzzy matching step does not produce effective enough results. That is why machine learning is further used. The data features for the machine learning model include:

1. **Actual Type** (*categorical feature*) - The type can be either software or hardware.

2. **Part** (*categorical feature*) - CPE part as either "a" for *application*, "o" for *operating system*, or "h" for *hardware*.

3. **CNA Source** (*categorical feature*) - The CVE Naming Authority is most closely associated with the vendor and product. Although the CNA does not currently factor much in the

prediction, there is a clear distinction in product naming among CNAs. We leave this to future research to determine if this should have more weight in the model's output.

4. **Word Vector Cosine Similarity** (*between [0, 1]*) - The cosine similarity between word vectors in the software inventory package name and the concatenated vendor and product from the CPE.

5. **Vendor Word Vector Cosine Similarity** (*between [0, 1]*) - The cosine similarity between word vectors in the software inventory package name and only the CPE vendor.

6. **Product Word Vector Cosine Similarity** (*between [0, 1]*) - The cosine similarity between word vectors in the software inventory package name and only the CPE product.

7. **Vendor Word Character Cosine Similarity** (*between [0, 1]*) - The cosine similarity between the word characters in the software inventory package name and the vendor.

8. **Product Word Character Cosine Similarity** (*between [0, 1]*) - The cosine similarity between the word characters in the software inventory package name and the product.

9. **Vendor Product Count** - The number of software products per vendor reported in the CPE. A vendor with a large product count has the risk of producing a false positive in the product matching.

10. **Vendor Vulnerability Count** - The historical number of vulnerabilities reported through the NVD for the given vendor. A high number of vulnerabilities increases the likelihood of a CPE match producing false positives. A low number of vulnerabilities indicates that products for this vulnerability can match at the less granular VENDOR level without much risk of false positives.

11. **Product Vulnerability Count** - Similar to the vendor vulnerability count, this is the count of vulnerabilities specific to the software product.

For each candidate CPE generated in the fuzzy matching step, the machine learning prediction outputs three classifications:

1. **Order** - An ordered set (HIGHEST, HIGH, MEDIUM, LOW, LOWEST, REJECT) on the confidence this target CPE is a good recommendation. The ordering classifier allows for a sorted presentation of recommendations back to the security operator. A *REJECT* output indicates the recommender should not present the option to the security operator.

2. **Level** - Either *VENDOR* or *PRODUCT*. If the target CPE went only to VENDOR for a software publisher with numerous products and vulnerabilities (e.g., Google, Microsoft, etc.), it would have many false positive vulnerability notifications. In contrast, a target CPE for a vendor with relatively few vulnerability notifications could safely match only to the vendor level.

Then the candidate CPEs are ordered based on Order classification first and then the Level classification. CPEs with a higher Order classification go before CPEs with a lower Order classification. When the Order classification is the same, CPEs with the Level classification of PRODUCT go before CPEs with the Level classification of VENDOR. Using the outputs of Order and Level, the recommender can present to human operators the most likely target CPEs first.

We generated training data samples using over 7,000 software product strings (i.e., software inventory package names) found on Wikipedia using the MediaWiki API [5] and Wikipedia categories related to software products. These names match against the CPE word vectors to produce 23,048 samples of training and test data for machine learning.

For each CPE in the NVD, the vendor and product strings receive pre-processing to remove non-alphanumeric characters. Words split based on the convention of using underscores and dashes to separate words in the CPE strings. Then, the word vectors are calculated for each vendor and product and stored in a hashed reference to the CPE. Training samples are pre-processed and formed into word vectors in the same way. The top 100 of the most similar CPE word vectors are

[5] https://www.mediawiki.org/wiki/API

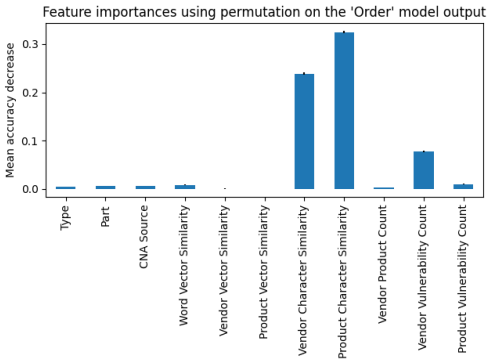

Figure 9: Feature Importance for 'Order' Output

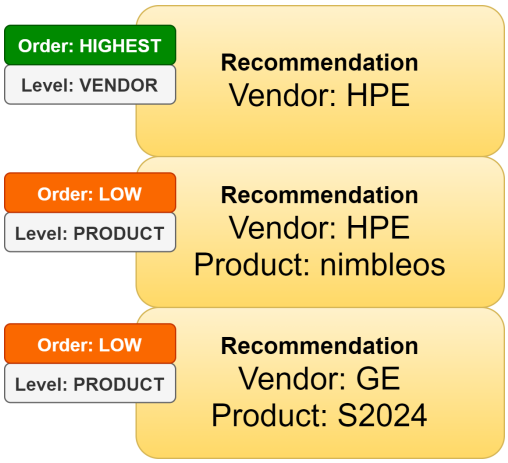

Figure 10: Example Recommendation Output

extracted as initial candidates using word vector cosine similarity tests on the product and vendor. Training data labels are generated based on manual inspection and general guidelines to avoid false positives. For instance, decisions to classify a match as a VENDOR typically only apply to vendors having less than 100 vulnerabilities per year. Otherwise, the match will likely produce far too many false positives.

The machine learning model uses a random forest classifier using a Gini impurity measure for the decision split, 100 decision trees in the forest, and an 80/20 training/testing split. Using a random forest classifier reduces the variance between the models and allows for more feature analysis. Table 6 shows the result for the *Order* classification. For the description of precision, recall, and F-score metrics, refer to the presentation of table 2. The highest and lowest order recommendations have very high F-scores because HIGHEST, LOWEST, and REJECT have more determinism and samples. The ordering in between (i.e., HIGH, MEDIUM, and LOW) requires more subjective judgment and more variance. Model classification performance for *Level* is almost 100% accurate. These classifications indicate high determinism in the output. The dataset used for training and testing is available on GitHub [6].

The primary output of the machine learning model is the *order* of presenting the results back

[6] https://github.com/pdhuff/cpe_recommender

to the security operator. Likewise, of the three outputs, *order* has the most number of options and highest variance. Figure 9 shows the feature importance for the *order* prediction using permutations as a measure. Each feature is arbitrarily modified ten times over the training and test data. The importance calculation, i, for feature j, is calculated as the difference in the overall model accuracy score, s and the mean permuted accuracy score:

$$i_j = s - \frac{1}{K}\sum_{j=1}^{K} s_{j_k} \tag{8}$$

The permuted feature measurement shows that the *Vendor Character Similarity* and *Product Character Similarity* have the highest importance in the model performance. Vulnerability count also has a significant impact on the model performance because our *order* labeling factored in the impact of choosing the wrong CPE. If a vendor or product has a higher vulnerability count, then the label is often ordered higher to minimize the potential for an operator missing applicable vulnerabilities. The lower feature importance for word vector similarity tests does not discount their use in the model. Indeed, word vector similarity is used to identify the input candidate CPEs for machine learning, but the variance in word vector similarity does not appear helpful in ordering.

Table 6: Machine Learning 'Order' Classification Results for CPE Matching

Recommended Order	Precision	Recall	F-Score	Support
HIGHEST	100%	98%	99%	937
HIGH	80%	66%	72%	232
MEDIUM	72%	46%	56%	211
LOW	89%	87%	88%	1,076
LOWEST	98%	99%	98%	5,854
REJECT	99%	100%	100%	14,738
Weighted Average	**98%**	**98%**	**98%**	**23,048**

3.5 Implementation and Evaluation

We tested the recommender system and compared it to a human analyst using i) 50 software inventory package names of commonly used software on the Microsoft Window (MS) platform, and ii)

50 hardware inventory names of commonly used network and computing hardware in an enterprise. The search involved inspecting these products on the NVD to find vulnerabilities associated with them, and validating the search result. Overall, the manual process took approximately 6.5 hours to identify target CPEs for the hardware and 70 minutes to identify target CPEs for the software.

Figure 10 provides examples for each prediction. Users have presented a reduced set of options priortized first by *Order* and then by *Level*. For example, if the machine learning finds a product match, the user first sees the product match before a less granular vendor match.

For software, the average number of manual search results was 119, and only 8% of those searches returned a conclusive result. In contrast, our recommender returned an average of 2 results with 40% conclusively identified, since it can accurately map software names to CPEs which in turn can accurately map to vulnerabilities. A sample of the results generated by the recommender is shown in figure 10.

For hardware, the average number of manual search results by the analyst was approximately 34 with only 28% returning a conclusive result, and the average number returned by the recommender was 2 with 48% returning a conclusive result. We consider no results returned as inconclusive. Since hardware is not as commonly represented in the NVD, no search results is not surprising with hardware assets.

For this small dataset of 100 hardware and software inventory package names, the recommender already saved over 7 hours and provided more accurate results. For larger systems with more hardware and software assets, the time saving will be even more. Furthermore, with the recommender automatically finding results, the cybersecurity operation has less risk of missing vulnerability notifications.

3.6 Conclusion

We have presented a new approach for organizations to automatically match their hardware and software inventory to CPEs used by standard vulnerability sources. The recommendation system outputs a small set of target CPE names for identifying applicable vulnerabilities based on the input

of software package inventory. Using a pipeline of NLP, fuzzy matching and machine learning, the system produces a much more accurate and useful result based on the ecosystem of software vendors and CVE Naming Authorities.

Our initial testing showed significant time savings, but more importantly, the results showed an improvement in accuracy. In future work, we plan to validate these results with a larger, more realistic dataset.

References

[2] Abdullah Abuhussein, Sajjan Shiva, and Frederick T Sheldon. "CSSR: cloud services security recommender". In: *2016 IEEE world congress on services (SERVICES)*. IEEE. 2016, pp. 48–55.

[12] Jesús Bobadilla et al. "Recommender systems survey". In: *Knowledge-based systems* 46 (2013), pp. 109–132.

[16] Robin Burke. "Hybrid recommender systems: Survey and experiments". In: *User modeling and user-adapted interaction* 12.4 (2002), pp. 331–370.

[21] Brant A Cheikes et al. *Common platform enumeration: Naming specification version 2.3*. US Department of Commerce, National Institute of Standards and Technology, 2011.

[33] Mark Dowd, John McDonald, and Justin Schuh. *The art of software security assessment: Identifying and preventing software vulnerabilities*. Pearson Education, 2006.

[37] Muriel Figueredo Franco, Bruno Rodrigues, and Burkhard Stiller. "MENTOR: the design and evaluation of a protection services recommender system". In: *2019 15th international conference on network and service management (CNSM)*. IEEE. 2019, pp. 1–7.

[39] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. "Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey". In: *ACM Computing Surveys (CSUR)* 50.4 (2017), pp. 1–36.

[46] Philip Huff and Qinghua Li. "Towards Automated Assessment of Vulnerability Exposures in Security Operations". In: *EAI International Conference on Security and Privacy in Communication Networks*. 2021.

[52] Andreas Kuehn and Milton Mueller. "Shifts in the cybersecurity paradigm: zero-day exploits, discourse, and emerging institutions". In: *Proceedings of the 2014 New Security Paradigms Workshop*. 2014, pp. 63–68.

[55] H. T. Le and P. K. K. Loh. "Using Natural Language Tool to Assist VPRG Automated Extraction from Textual Vulnerability Description". In: *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*. Mar. 2011.

[56] Triet Huynh Minh Le, Bushra Sabir, and Muhammad Ali Babar. “Automated software vulnerability assessment with concept drift”. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019, pp. 371–382.

[60] Kylie McClanahan and Qinghua Li. “Automatically Locating Mitigation Information for Security Vulnerabilities”. In: *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids*. 2020.

[62] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[66] Fitzroy D Nembhard, Marco M Carvalho, and Thomas C Eskridge. “Towards the application of recommender systems to secure coding”. In: *EURASIP Journal on Information Security* 2019.1 (2019), pp. 1–24.

[71] Paul Resnick and Hal R Varian. “Recommender systems”. In: *Communications of the ACM* 40.3 (1997), pp. 56–58.

[73] Jukka Ruohonen. “A look at the time delays in CVSS vulnerability scoring”. In: *Applied Computing and Informatics* 15.2 (2019), pp. 129–135.

[74] Ernesto Rosario Russo et al. “Summarizing vulnerabilities’ descriptions to support experts during vulnerability assessment activities”. In: *Journal of Systems and Software* 156 (2019), pp. 84–99.

[87] Peichao Wang et al. “Intelligent Prediction of Vulnerability Severity Level Based on Text Mining and XGBboost”. In: *2019 Eleventh International Conference on Advanced Computational Intelligence (ICACI)*. 2019, pp. 72–77.

[92] Y. Yamamoto, D. Miyamoto, and M. Nakayama. “Text-Mining Approach for Estimating Vulnerability Score”. In: *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. Nov. 2015, pp. 67–73.

[94] Fengli Zhang and Qinghua Li. “Dynamic Risk-Aware Patch Scheduling”. In: *IEEE Conference on Communications and Network Security (CNS)*. 2020.

[95] Fengli Zhang and Qinghua Li. “Security Vulnerability and Patch Management in Electric Utilities: A Data-Driven Analysis”. In: *The 1st Radical and Experiential Security Workshop (RESEC)*. 2018.

[96] Fengli Zhang et al. “A Machine Learning-based Approach for Automated Vulnerability Remediation Analysis”. In: *IEEE Conference on Communications and Network Security (CNS)*. 2020.

4 Cyber Threat Intelligence Exchange

Adversaries have the upper hand in cyber attacks. They benefit from anonymity, both in person and in purpose. In contrast, targeted entities (e.g., companies) have difficulty distinguishing everyday benign activities from malicious activities. Thus, entities spend prodigious efforts to gain actionable threat intelligence. In a recent survey on Cyber Threat Intelligence (CTI) sharing, security professionals strongly agree that intelligence sharing supports breach detection/recovery and vulnerability identification/mitigation efforts [97]. However, many technical, trust, legal and cultural barriers prohibit more widespread threat information sharing [31].

Many cyber threats target critical infrastructures in the private sector. These target entities have the same trust barriers and even more technical and legal barriers due to the limits of qualified security professionals working at each organization. A recent report on cyber threat sharing indicated only 3% of private sector participants shared any threat indicators in 2018 [48].

Furthermore, the value received from CTI is often lacking due to various technical challenges and missing context. In one study [83] 70% of respondents find shared threat data too voluminous and complex for actionable intelligence. Similarly, [35] finds CTI solutions need to enhance their ability to provide context and flexibility to improve the overall value proposition.

4.0.1 The Current State of Threat Sharing

Organizations are rapidly developing the competency and appetite to participate more in threat-sharing communities. The global rise in security operations centers, through which most CTI exchange occurs, has an expected market growth of 11.5% through 2025 [41]. However, with current approaches heavily focused on classified data and government intelligence services, actionable data is too little and too late. Likewise, as [32] points out, private sector organizations have little motivation to share their threat data sustainably.

Entities share threat data to gain a better understanding of the risk posed to their mission. An average entity may experience tens of thousands of malicious probes from the Internet per day. However, most probes result from automated scanning and do not represent a motivated and

intelligent human adversary. Entities participate in threat sharing to distinguish actual danger from benign in hopes of mitigating the threat before it manifests.

Society has an interest in preventing cyber threats from entities that provide critical services and infrastructure. Military and law enforcement agencies would generally provide protection, but they have limited purview into the interaction between adversaries and private entities. Government agencies, national Computer Emergency Response Teams (CERTs), and non-profit Information Sharing and Analysis Centers (ISACs) offer two-way threat-sharing services to address this gap.

However, private entities have many barriers encumbering CTI sharing. A private entity wishing to share threat information risks attribution of the cyber threat, resulting in costly legal disputes, regulatory investigation, and reputational damage. For example, a mistaken analysis of VPN logs to maintain a failed water pump led to a federal investigation of cyber warfare [93]. In [49], legal compliance and limiting attribution are identified as the primary challenges for organizations wishing to share their own CTI with others.

Additional barriers exist with sharing of classified intelligence to private entities. Programs exist to clear private sector entities, but they come at a high cost. Then, moving classified intelligence to actionable threat and vulnerability mitigation cannot keep pace with adversarial intrusion techniques' dynamic nature. Likewise, attempts for fully bidirectional threat sharing have mostly failed.

4.0.2 Contributions

This chapter provides a solution for entities to share observed CTI without attribution using a permissioned blockchain. We propose a novel approach to a Distributed Anonymous Payment (DAP) scheme [75] for permissioned blockchains to allow for anonymous transactions in CTI sharing. This solution also efficiently maintains anonymous authentication and provides revocation services for entities. It does so by splitting maintenance of the Merkle tree used for anonymous authentication between participating peers, which allows for more regular updates of the Merkle Tree across the distributed ledger.

Anonymous transactions address the legal and regulatory barriers organizations have with cyber threat attribution, increasing CTI sharing on the ledger. We then propose a new chaincode to incentivize CTI creation for the cooperative benefit of participating entities. The chaincode targets the barriers preventing bidirectional threat sharing between private sector entities and government agencies by generating timely and actionable CTI without the need for costly declassification.

The chaincode also seeks to reduce volume and increase value in CTI. Human analysts control the volume of threat data through work evaluation functions. Whereas automated log sharing solutions produce data at the speed of machines, the chaincode produces intelligence at the speed of humans. Furthermore, human analysts should find the intelligence actionable because the chaincode originates directly from private entity queries.

4.0.3 Organization

Section 4.1 reviews related work. Section 4.2 introduces the building blocks for our approach. Sections 4.3, 4.4, and 4.5 presents the proposed approach and its major components. Section 4.6 discusses evaluation results. Section 4.7 concludes this chapter.

4.1 Background and Related Work

CTI exchange programs fall into three categories:

1. Classified Threat Sharing - Provides automated classified threat indicators to its members. The DHS Enhanced Cybersecurity Services (ECS) is an example of this type of service[34].

2. Data Lakes - Collects a large volume of logs from its members and centrally analyzes the data. The Department of Energy Cyber Risk Information Sharing Program (CRISP) uses the data lake model [29].

3. Analyst to Analyst - Threat hunting analysts exchange data over a shared platform. The European Union Agency for Cybersecurity recommends the Malware Information Sharing Platform for community threat sharing [35].

This chapter targets the third category of CTI in which human analysts directly share threat intelligence and indicators between entities. The most commonly shared threat data includes low-level indicators such as IP addresses, URIs, DNS names, and file hashes collected automatically or via threat hunting. Our platform supports sharing of other security information as well, e.g., vulnerability mitigation information. Many services provide one-way data sharing to the entity of known malicious threat indicators.

Stillions' Detection Maturity Levels [80] characterizes this type of data as lower-level evidence of an intrusion attempt. In contrast, higher levels of intelligence include data about how the adversary operates and their motivations.

The work of creating CTI involves tying lower-level indicators to adversarial motivation. However, these indicators exist in the networks of private entities and outside of the direct purview of CTI producers. Timely bidirectional CTI exchange means indicators and resulting CTI are shared freely. The producers receive value by better tracking malicious activity, and consumers receive value through an improved understanding of adversarial risk.

Using a distributed ledger, we can commoditize CTI work as described in section 4.5 while, at the same time, eliminating trust barriers that preclude the sharing of threat indicators.

4.1.1 Blockchain Technologies

The permissioned ledger fundamentally uses blockchain as a basis for distributed trust. Blockchain has gained popularity with cryptocurrency technologies like bitcoin [63], and ethereum [89] making possible public distributed transactions with no central authority. Several recent works have suggested using blockchain technologies for CTI exchange [72, 42, 44]. Our work differs by addressing attribution and targeting CTI sharing communities of trust through a permissioned ledger.

The use of a permissioned blockchain presented in [5] has growing acceptance as a general-purpose distributed ledger. While still public, in the sense of being accessible over the Internet, permissioned blockchains take advantage of partial trust relationships in a system. In the Hyperledger Fabric project, network peers first execute transactions and then order and distribute them

onto the blockchain. This approach allows for more complex transactions because peers can detect state and denial of service problems before the chaining operation.

We choose a permissionless blockchain over a public blockchain because of privacy considerations. A CTI sharing community is often open only to participating members from a given sector or nation-state. Although peer entities have no problems with attribution among the community, privacy concerns would likely arise in a public blockchain.

4.1.2 Zero Knowledge Proofs

The public nature of blockchain systems spotlights the need for anonymity and private information retrieval. Common to most solutions to these problems are zero-knowledge proofs (ZKP), which allow authentication without identification. In [20], Chaum first developed an e-cash system in which a user could present proof of authentication from some certifying entity without revealing the user. Pseudonym systems in [57] have a similar mechanism to allow entities to operate under a pseudonym untraceable to their original authenticated identity and ultimately form a chain of pseudonyms to conduct anonymous transactions in a system.

Direct Anonymous Authentication (DAA) systems extend and implement ZKP and have widely deployed on trusted platform modules (TPM), and blockchain systems [15, 19, 18, 17]. Most recently, the anonymizing idemix library has become available as a core service in Hyperledger Fabric.

However, DAA schemes do not have a mechanism for incentives, and they require additional roles in managing access to the ledger. Instead, we look to recent advancements in cryptocurrency. The explosive growth of cryptocurrencies has ushered in a wave of innovation in anonymizing transactions in the past decade. Anonymous spending in cryptocurrency is made possible through zero-knowledge Succinct Non-Interactive ARgument of Knowledge (zk-SNARKS) presented in [30]. Zerocoin [61] is one of the first systems proposed to support anonymous transactions on top of bitcoin. Zerocash [75] and others [51] made use of zk-SNARKS to make this more feasible and extend the system to prevent tracing the history of a coin and improve efficiency.

Although permissioned blockchains do not require a cryptocurrency incentive, we propose an incentive mechanism for the desired outcome of high quantity and quality threat data. The "gas" or currency of cybersecurity exists in human work and actionable CTI.

4.2 Building Blocks

Before presenting the approach to non-attributable CTI sharing, we introduce the building blocks used by our approach.

4.2.1 Sparse Merkle Trees

Merkle trees provide an efficient data structure to authenticate information. They are used on the blockchain to verify transactional integrity. Branches of the tree get formed from the combined secure hashes of its children. In this way, anyone can verify the membership of a tree leaf by comparing the calculated Merkle root with some other valid Merkle root.

Sparse Merkle Trees make use of the property that the path to any given leaf is a function of a small number of branches up to the Merkle root. In the example shown in Figure 11, we store a minted coin, `cm`, as a leaf in the Merkle Tree. The leaf's index is determined by the branch direction down the tree, in which a 0 means the left branch, and a 1 means the right branch. Then, for someone to later validate the inclusion of `cm`, they need only the index and the tree branches along the path indicated by the index, which is necessary to calculate the root.

We represent the tree path as `path`, which contains attributes for the index location in the tree, $\texttt{path}.addr$, and the branch siblings, $\texttt{path}.S$ necessary to calculate the Merkle root.

4.2.2 Distributed Anonymous Payment

First, distributed anonymous payment (DAP) schemes allow an entity to prove they have an electronically minted coin, `cm`, without actually revealing the coin. The proof also requires the entity to provide knowledge of an associated, yet untraceable, serial number, `sn`, to prevent an entity from double-spending.

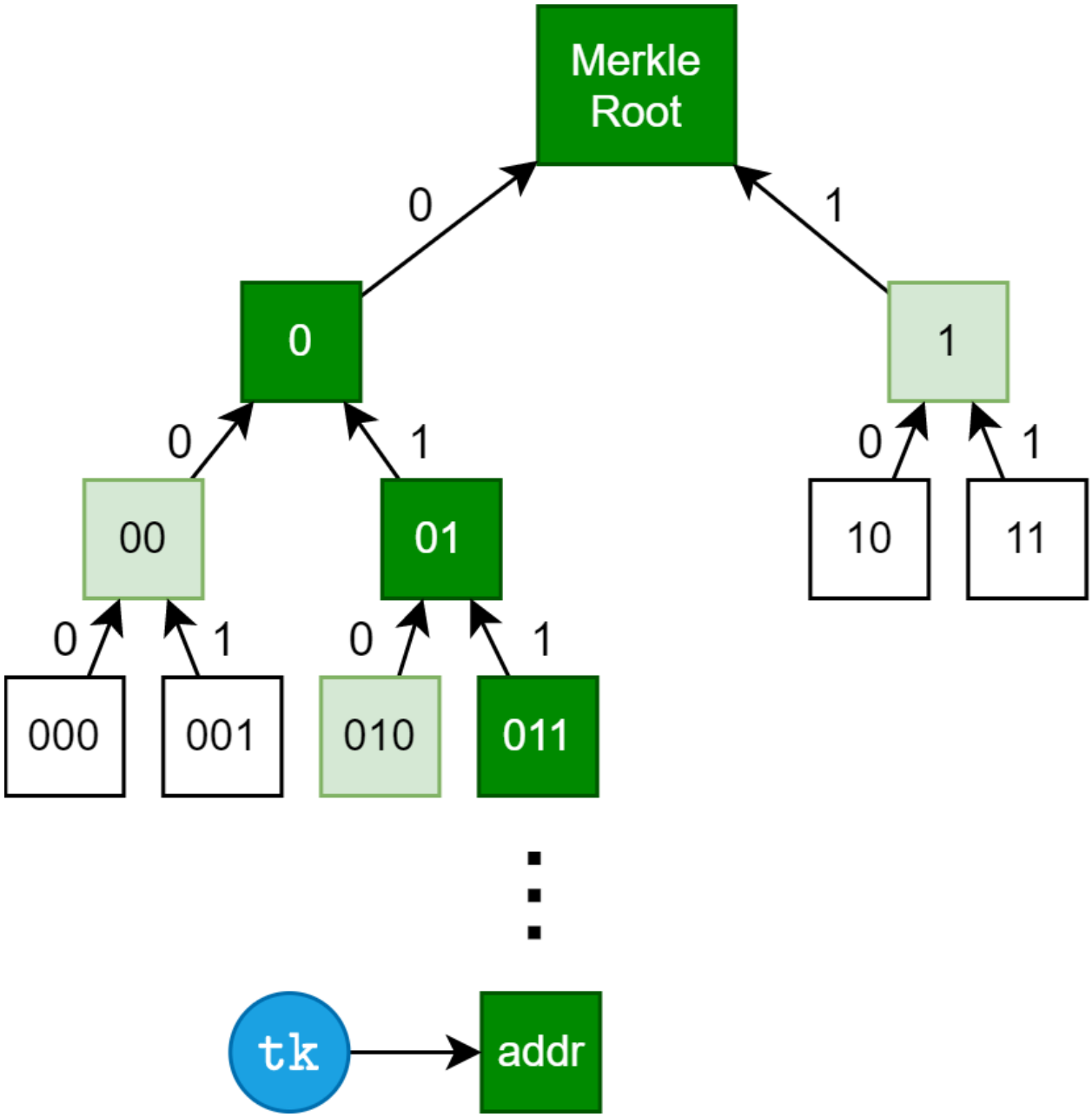

Figure 11: Sparse Merkle Tree.

DAP schemes have the important property of retaining the minted coin as a valid leaf value in the Merkle Tree. Unlike Bitcoin, they do not have the luxury of maintaining an unspent transaction object (UTXO) inventory. To do so requires identifying spent coins, which DAP schemes do not reveal. Therefore, we must evaluate the Merkle tree size appropriate to support the life of the blockchain.

4.2.3 zk-SNARKs

The proof of knowledge in [75] uses zero-knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARK) proofs from [11]. zk-SNARKs provide an efficient proof construct and verification mechanism. Our proof demonstrates the knowledge of a `cm` $\in$ `CMList` without revealing `cm`, which equates to an anonymous user proving, "I have a valid token, but to ensure my anonymity, I am not going to tell you which token."

At its core, a zk-SNARK equates to demonstrating knowledge of a well-formed polynomial, $p(x)$, such that $h(x)t(x) = p(x)$, where $t(x)$ is a target polynomial available to the ledger, and $h(x)$ is derived by the prover as $h(x) = p(x)/t(x)$. The prover constructs the polynomial, $p(x)$, through an algebraic circuit available on the ledger which has been translated from code representing the Merkle Tree proof of knowledge. The prover samples some arbitrarily chosen secret s, such that $h(s)t(s) = p(s)$. To ensure the integrity of the target polynomial and sampled value, s, all operations are performed using homomorphic encryption with generator, g, such that $(g^{h(s)})^{t(s)} = g^{p(s)}$.

The process for non-interactive proof and verification consists of the following steps:

1. **Multi-Party Setup** - A multi-party setup protocol occurs to produce the public parameters, pp, which includes the homomorphic encryption of the powers of x in the secret polynomial of dimension, d for secret, s. Thus, the proving key consists of the powers necessary to compute the secret polynomial, the target polynomial, and sampled values to ensure zero knowledge of the secret polynomial. An initial setup requires multiple parties with strong zero-knowledge guarantees [13]. The keys used for proving and verification are referred to as the *common reference string*.

2. **Algebraic Circuit** - A program to construct the zero-knowledge proof converts to an algebraic circuit by flattening the program into a series of expressions in the form $x = y \texttt{op} z$, which form the so-called circuit wires. Ultimately, these form the basis of the secret polynomial coefficients. In our case, the circuit consists of the Merkle Tree proof of inclusion.

3. **Proof** - An entity constructs a proof of knowledge demonstrating they have a valid token in the Merkle tree using both the public parameters and algebraic circuit. The proof is non-interactive because the prover does not need to exchange keys to produce the proof statement. Zero-knowledge comes through a key sampled by the prover, which conceals the secret polynomial.

4. **Verification** - Verification is performed in the chaincode of the ledger to ensure the construction of the secret polynomial in addition to the public inputs to the circuit is valid.

Besides the original works in zk-SNARKs, the papers [69, 9] provide good tutorials on the process.

4.3 Distributed Ledger for Threat Sharing

Distributed ledgers provide transactional integrity for large and diverse communities. In its most well-known cryptocurrency implementations, distributed ledgers supply a high assurance system for transacting digital goods such as Bitcoin. Our scheme considers human work as the exchanged commodity for cybersecurity threat sharing. The work of threat identification and attribution involves costly human labor to identify artifacts, piece together the adversarial objective, and tie cyber observables to malware campaigns and threat actors. Entities receive value through more actionable intelligence and an improved understanding of cyber risk.

The use of a distributed ledger for cybersecurity work is not without precedent. [77] proposes the use of economic incentives to incentivize secure data sharing. Also, in many ways, a marketplace for threat information can be compared to software bug bounty programs where companies wishing to fix software vulnerabilities before an adversary exploits them monetize the work of

finding vulnerabilities [43]. However, with cyber threats, the work production comes from entities wishing to protect their systems better.

We propose a distributed ledger in which any participating entity submits monetized threat intelligence work in the form of structured work queries as transactions on the ledger. Entities requesting work do so through anonymous credentials using a web application tied to a peer entity on the distributed ledger. Participants use the same web application to search for information about a given threat. The ledger does not record searches as transactions.

4.3.1 Distributed Ledger Network

This section proposes a permissioned blockchain network architecture to support the exchange of threat intelligence between participating entities. Our implementation for threat sharing uses a permissioned blockchain. These differ from public blockchains by requiring authenticated access and eliminating the need for proof-of-work or proof-of-stake consensus. Chaincode is a set of smart contracts installed by participating entities and serves as the blockchain's central service rather than the currency transaction object. With cryptocurrency, smart contracts are a service of the blockchain, but with permissioned blockchains, the blockchain is a service of the smart contract.

Also, cryptocurrencies overcome almost all trust boundaries, but this is not always desirable, especially with CTI. Instead, we use the permissioned blockchain to overcome trust boundaries existing between organizations.

Figure 12 shows an example blockchain network in where the shaded area represents elements required by the blockchain and users involved in CTI access the network outside of the shaded area. Fundamentally, the blockchain includes a group of entities, referred to as peers, who have consensus on the chaincode execution and maintain a copy of both the blockchain and the current state database of chaincode assets (or objects).

Peers join the network either initially or through peer consensus. The collective peers comprise the distributed system's nodes, and they participate in the validation of new blocks and storage of

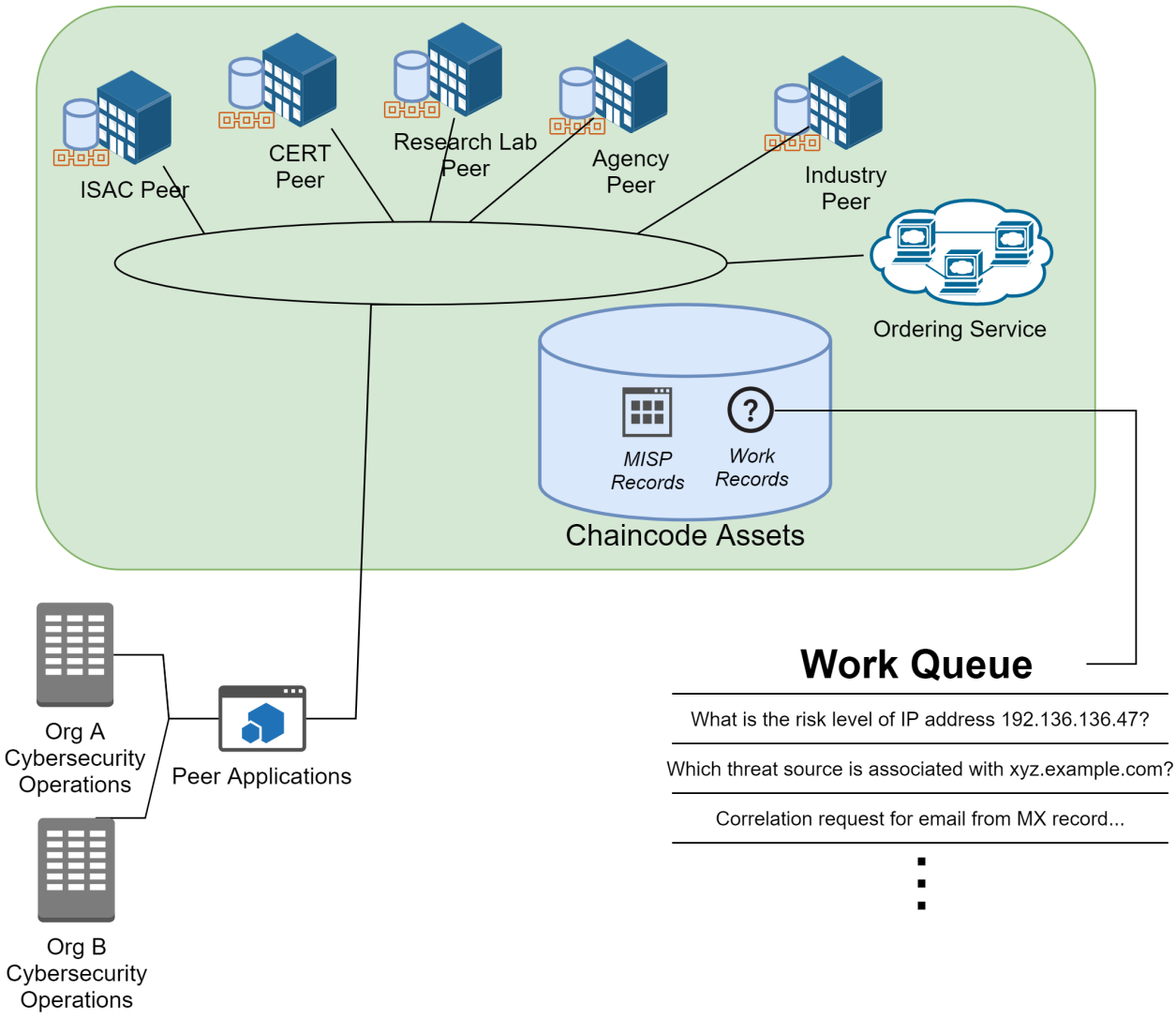

Figure 12: Threat Ledger Network.

the data. However, with permissioned blockchains, peers also provide the service of user interaction with the blockchain network.

An organization does not need to be a peer of the blockchain to participate in the service. Instead, peers provide credentialing services through their certificate authority. Users of other organizations are then permitted to execute chaincode transactions through peer applications.

In the example shown in Figure 12, the peers include organizations typically involved in threat sharing, such as government agencies, CERTs, ISACs, and research labs. These organizations have the incentive and resources to install and maintain the peer service needed for threat sharing. If a private entity wanted to participate in the network, they would only need to obtain credentials from a peer and use a published web application, thus, significantly lowering the bar of complexity for threat exchange participation.

The network also requires an ordering service. Blocks of transactions get added to the blockchain through the ordering service. Consistent with the execute-order-validate consensus approach described in [5], peers will first simulate the execution of proposed transactions before sending them to the ordering service. The ordering service then packages valid transactions into the next block and sends them to all network peers.

4.3.2 Chaincode Assets

The network's chaincode centers on CTI reports commonly exchanged between organizations. We choose to use the standard MISP format [85]. Other CTI taxonomies include STIX [10], and the Common Cyber Threat Framework [1], but the MISP format is extensible and concentrates on the threat report instead of the observable artifacts. By aggregating artifacts into event reports, we can more easily form a high-level representation of the CTI report's value.

An sample MISP report with object relationships is shown in figure 13. A MISP *Event Report* contains the creating organization (or anonymous), description, and report object, which can range from single threat observation reports to several thousand indicators and sightings of a malware family. There are over 200 open object definitions, and reports can contain multiple objects. Tags

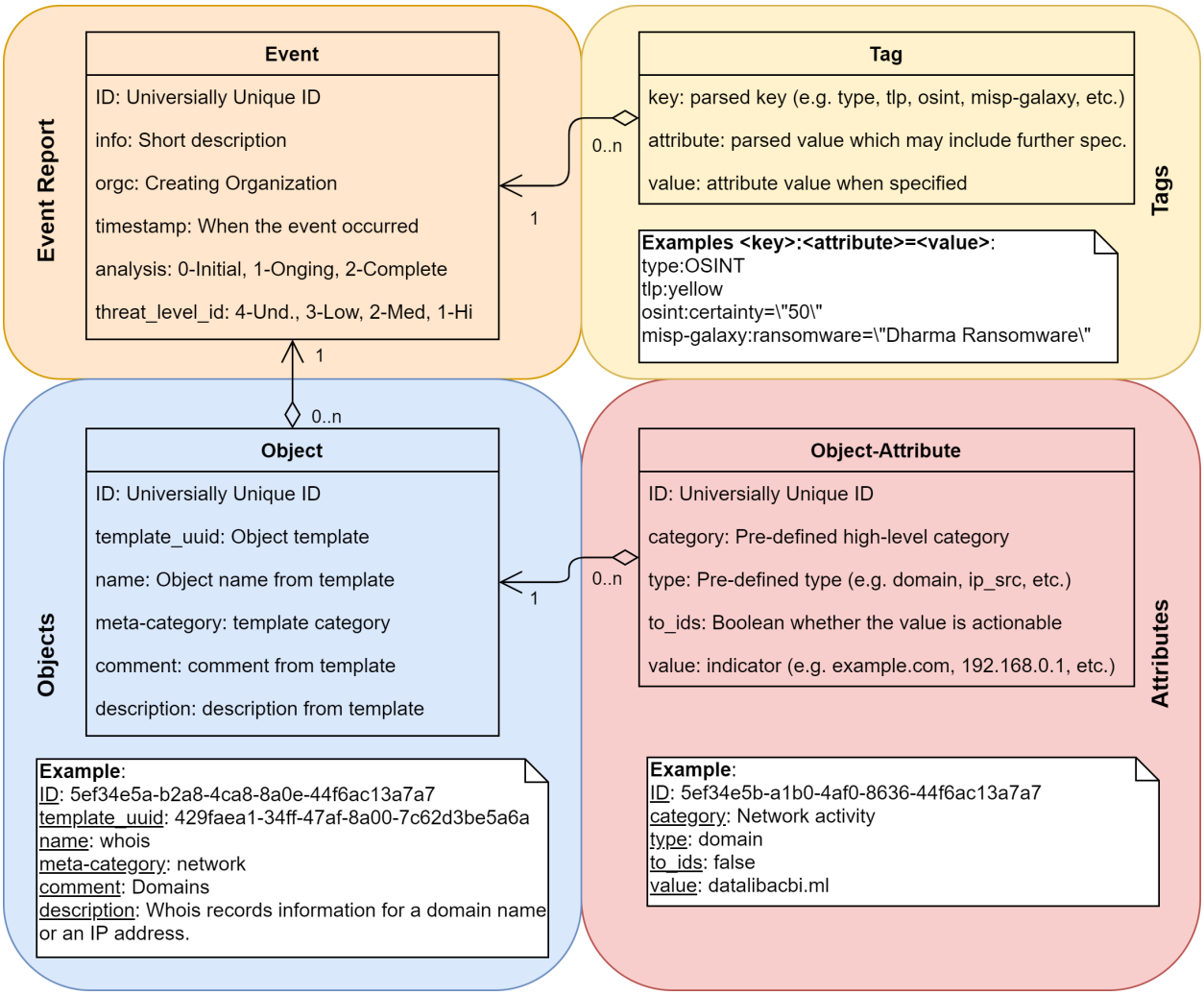

Figure 13: MISP Data Object Model.

describe the report in terms of the information-sharing community. Example tags include the DHS Traffic Light Protocol, malware classifications, IDS rules, and admiralty scale. The tags can be helpful in chaincode for defining access control rules, expiration, and other state transition logic.

Finally, object attributes tie to the reported objects and contain the observable artifacts associated with an event, such as IP addresses, URIs, file hashes, and email addresses. Attributes are the primary search targets for the network. Each network peer stores a document-oriented NoSQL database of existing reports and indexes the attributes for fast searching and correlation.

Besides the threat objects, we also define two assets used for managing the quality of threat

reports. The *work* asset represents human work and consists of structures for both the problem and the solution. When first submitted, the solution is empty and queued for human analysis. Examples of work may include associating tactics, techniques, and procedures (TTP) to threat artifacts or attribution of a threat report. Other types of work might include validation or annotation of reports to assist in automatic classification, and generation of mitigation actions for vulnerabilities that the adversary tries to exploit.

Finally, a *tree* asset serves to facilitate anonymous authentication and manage the human work by controlling the input, incentivizing the output, and anonymizing the submittal of software artifacts.

4.4 Non-Attributable Token Authentication

For the CTI distributed ledger to function, we must provide its users with anonymization guarantees. We now present the approach for anonymous authentication using a Merkle Tree for zero-knowledge commitment. To start, we present the process of token commitment. Then we show an approach of splitting the tree to support more authentication features such as revocation and value-based spending.

4.4.1 Anonymous Token Spending

A user receives a token upon the chain code validating some threat intelligence work, or perhaps as part of some bootstrapping process where new users have a limited set of tokens. A user will provide a token to the ledger when performing work for the chaincode to later validate. Then, once the chaincode validates the token, commitment occurs by adding the token as a leaf to the Merkle tree, `tree`.

The user arbitrarily samples a secret key through the security parameter, λ representing the key length and pseudorandom function $\texttt{Gen}(1^\lambda)$. A user may safely use the secret key repeatedly as a witness to multiple tokens. For each new token, a user arbitrarily generates a serial number, `sn`, in the same way. Then, using a collision-resistant hash function, $\texttt{CRH} = (0,1)^* \rightarrow 0,1^\lambda$, the token is

generated as shown in the following functions.

$$\begin{array}{ll} 1: & \mathtt{sk} \leftarrow \mathtt{Gen}(1^{\lambda}) \\ 2: & \mathtt{sn} \leftarrow \mathtt{Gen}(1^{\lambda}) \\ 3: & \mathtt{tk} \leftarrow \mathtt{CRHsksn} \end{array}$$

The sparse Merkle tree then gets calculated with the inclusion of the token as $(\mathtt{rt}, \mathtt{path})$. The user then has the following public and private data related to the token.

$$\begin{array}{ll} 1: & \mathtt{tk}_{\mathtt{pub}} \leftarrow (\mathtt{rt}, \mathtt{sn}) \\ 2: & \mathtt{tk}_{\mathtt{pri}} \leftarrow (\mathtt{sk}, \mathtt{path}) \end{array}$$

Algorithm 1 Token Verifier Circuit

Public Parameters: pp
Public Input: rt, sn
Witness: sk, path
Output: π - proof of inclusion

```
1: procedure TOKEN_VERIFIER
2:     tk ← CRHsksn
3:     rt_tk ← the smt calculation using tk and path
4:     if rt = rt_tk then
5:         return true
6:     else
7:         return false
8:     end if
9: end procedure
```

Algorithm 1 shows the zk-SNARK circuit for proof and verification. To generate a zk-SNARK proof, a user supplies the public parameters, pp, which includes the common reference string for proving and the zk-SNARK circuit. Public input includes both the Merkle root, rt, demonstrating knowledge of a valid token, and the serial number, sn, formed through the witness. The witness includes the secret key, sk, and the path down the tree to the token.

The chaincode on the distributed ledger verifies the proof represented in algorithm 2. Here, the public parameters, pp, include the portion of the common reference string used for verification in the ledger. The verification includes (i) checking to ensure the zero-knowledge proof is valid,

(ii) verifying the Merkle Root is a valid root for the ledger, and (iii) the serial number represents an unspent coin. The first check uses the zk-SNARK for the network. For the second check, the ledger must include a set of valid roots, and we describe this process in section 4.4.2. The final check on whether sn exists in SNList prevents a double spend.

Algorithm 2 Verify Token Proof

Public Parameters: pp
Input: π, rt, sn
Output: Valid or Invalid

1: **procedure** VERIFY_PROOF
2: $valid \leftarrow verify(\texttt{pp}, \pi, \texttt{rt}, \texttt{sn})$ ▷ zkSNARK verification
3: **if** $valid \wedge \texttt{rt} \in \texttt{RTList} \wedge \texttt{sn} \notin \texttt{SNList}$ **then**
4: **return** Valid
5: **else**
6: **return** Invalid
7: **end if**
8: **end procedure**

4.4.2 Merkle Tree Structure and Root Updates

In the token spending scheme described above, a root update when inserting a batch of new tokens to the tree would make token spending attribution trivial. An entity would only need to search the ledger for the root associated with a token proof to identify the user.

To prevent this attack, we designate an entity to perform the service of sending out root updates at a time interval, t_{new}. Then validation should only include roots published within some time interval, t_{expiry}. Thus, a user wishing to spend a token must wait within a timespan of t_{new} after receiving the validation. Also, a token proof will be valid within a timespan of t_{expiry} from the proof construction. The expiration prevents token attribution because the prover supplies only recent token roots instead of the root calculated at token insertion.

The problem then becomes regularly distributing the tree paths to the network, which we now address. A Merkle tree in a DAP scheme may have token leaves distributed in any order. The location of the leaf in the tree has no association with the identity of the token owner. However,

permissioned blockchains have inherent organizational structures, which the ledger can use for more robust authentication features and storage efficiency.

For a Merkle tree of height, h, the branch levels are split into three levels, h_{net}, h_{org}, and h_{user} as shown in figure 14. Thus, the tree supports $2^{h_{net}}$ organizations and each organization may have $2^{h_{org}}$ users.

By dividing the tree height, we minimize the size of tree updates and storage requirements to only those necessary for the entity's role in the network. As an example, a tree with a height of 32 requires approximately 256 GiB of storage. Also, to keep the siblings, $\texttt{path}.S$, up to date, the network must distribute a similar-sized update. However, using a permissioned blockchain's organizational structure and setting the h_{net} level at 14, the network updates only require 1 MiB while allowing for 2^{14} organizations.

Each organization is responsible for maintaining its similarly sized sub-tree to distribute $\texttt{path}.S$ updates to its users.

The organizational tree structure supports several other services, which we now describe.

4.4.3 Revocation of Anonymous Authentication Tokens

Any network peer entity or organization may wish to revoke tokens as users leave, tokens become compromised, or users abuse the network. Since the tree divides into sub-trees of organizations and users, such revocation becomes trivial. User tokens are revoked by setting the desired token sub-tree to null and recalculating the root. Similarly, the network could revoke entire organizations by setting the organization sub-tree to null.

The revocation scheme works because tokens are not anonymous, and the Merkle tree does not need to hide the token holders' identities. A token spend only reveals the serial number, which cannot associate with the token. The network may safely maintain an identity on the token tree while preserving non-attribution in token spending.

The revocation latency ties to the t_{expiry} time interval associated with root updates. Attempts to authenticate using a revoked token will guarantee to fail after t_{expiry} because the proof of token

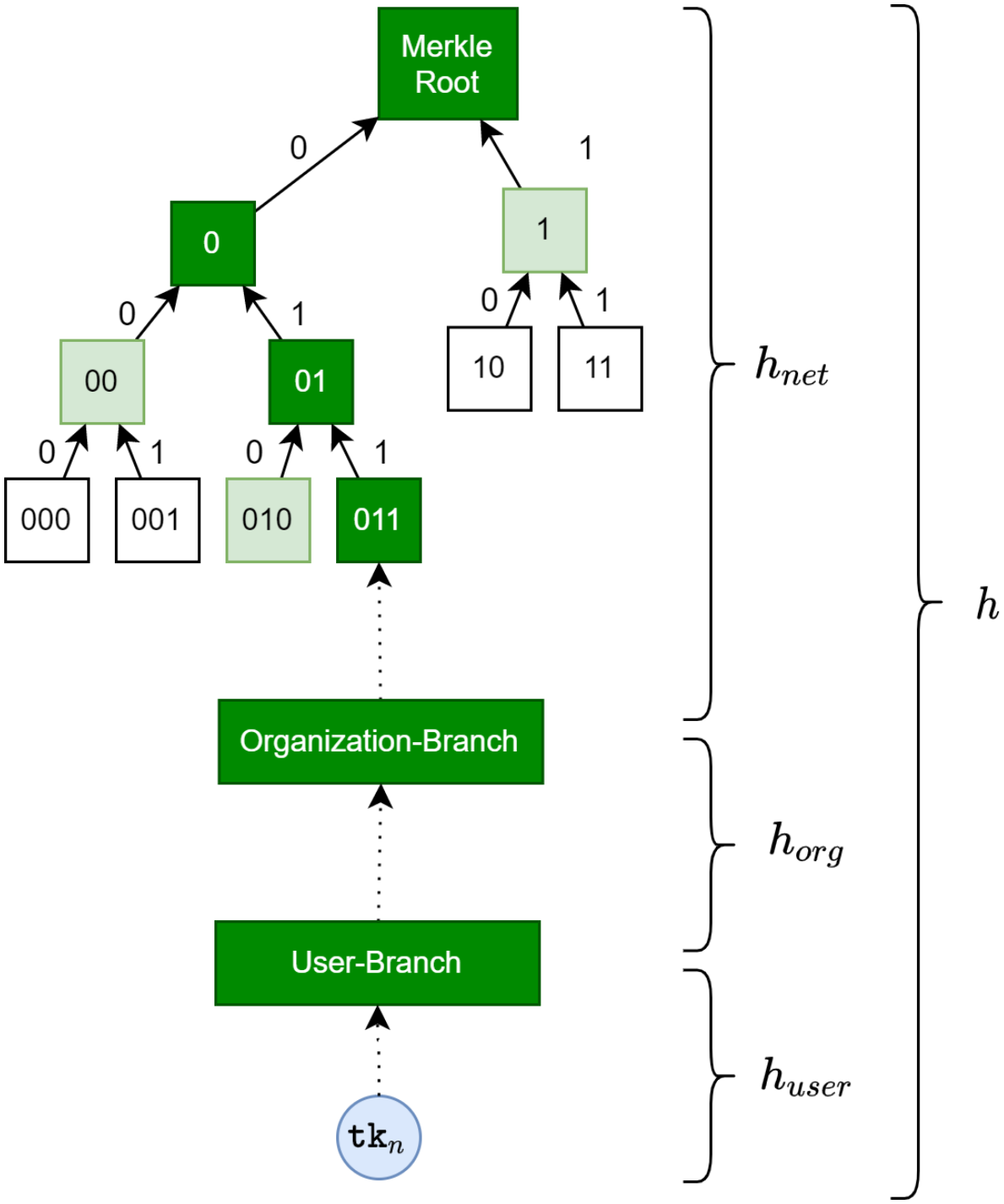

Figure 14: Merkle Tree Structure.

inclusion no longer works with the new root.

4.4.4 Adding Value to Tokens

In a cryptocurrency, value is an attribute of the coin itself, and spend operations *pour* an old set of coins into a new set with the same value preserved. However, pouring coin value creates problems in the proposed scheme because the primary purpose of the token is for non-attributable authentication, and supporting a large number of token spends adds unnecessary complexity.

Instead, we propose tokens only have a value of 1, and we increase the user Merkle tree height to support a large number of tokens. Only the user only needs to maintain the path siblings for any levels below h_{org}. Knowing these paths allows the user to construct a valid proof without the network or organization having to maintain a tree height to support a large number of possible tokens.

For example, if the network maintained a tree height of 14 at 1 MiB, and each organization maintained a sub-tree height of 18 at 8 MiB, each user could maintain their sparse sub-tree of 64 levels to support a vast number of tokens far beyond the maximum necessary.

4.4.5 Authentication without Spending

Finally, by maintaining tokens with revocation services, they provide a useful means of anonymous authentication. There are several scenarios where users might desire anonymity. A user may wish to perform an anonymous search on the network, e.g., searching for a particular IP address. Performing such a search could infer the organization's attribution as a victim of the malware.

To support anonymous authentication only, we make a minor modification to the token spending circuit and remove the serial number as the public verification parameter. Additionally, we hash the timestamp, ts, with the root to prevent replay attacks.

Algorithm 3 Token Authentication Circuit

Public Input: $\texttt{pp}, \texttt{rt}, ts$
Witness: $\texttt{tk}, \texttt{path}$
Output: Whether the calculated root matches the given root

procedure TOKEN_AUTH
    $\texttt{rt}_{\texttt{tk}} \leftarrow$ the smt calculation using tk and path
    if $\texttt{CRHrt}ts = \texttt{CRHrt}_{\texttt{tk}}ts$ **then**
        return true
    else
        return false
    end if
end procedure

4.5 Chaincode for CTI Work

This section presents in detail the state program model used for managing work on the network. Several peer-authenticated transactions occur to update threat reports, which we do not formalize. The transactional updates to threat reports are essential but straightforward. Instead, we focus on the *Work* asset transactions to facilitate the expansion of threat knowledge and automation beyond existing services. Recall that a *Work* asset consists of problem and solution data structure which maps to an *Event Report* asset.

Work asset transactions focus both on the problem of submitting CTI anonymously and on validating the quality of the CTI. The cybersecurity community has not extensively considered the use of non-attributable CTI, and the chaincode recognizes this by including a set of evaluation states.

Figure 15 shows a state transition diagram of the workflow from the addition of *Work* to the completion of a solution. Each state transition represents a chaincode function made available to the network for processing the ledger. The ledger must maintain state to support asynchronous processing of transactions and high assurance in the logic of the chaincode.

The object variables for the state program include the following chaincode assets:

$$Var = \{\texttt{event\_record}, \texttt{work}, \texttt{token\_tree}\} \tag{9}$$

A `threat_record` asset includes the complex data structure represented in figure 13 and described in section 4.3.2. Assets for `work` have a problem/solution data structure that stores the proposed problem and maintains a set of proposed solutions for evaluation. The tree asset supports the use of tokens described in section 4.4.

The program graph over Var is defined as

Definition 4.1. *State Transaction Program Graph*

- S - Set of states
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$ - Transition effect function.
- $R \subseteq S \times Cond(Var) \times Act \times S$ - Conditional transition relation
- $S_0 \subseteq S$ - The set of initial states
- $g_0 \in Cond(Var)$ - The initial condition

The function $Eval$ comprises the set of evaluations over Var, and the function $Cond$ comprises the set of conditional expressions over Var.

Work state is maintained through the smart contract logic. Valid work states include $S =$ `READY_WORK`, `IN_PROGRESS`, `READY_EVAL`, `IN_EVAL`, `ADD_WORK`.

Anyone with a valid token may submit a `work` record to the network accompanied with a token proof. The chaincode first evaluates the token proof as a guard condition for the work queue. In this way, the `work` has no attribution to an entity, but the entity authenticates as a valid user of the network. Also, the ledger preserves the quality of the work queue by requiring an entity to give up something of value in exchange for work performed.

Each `work` asset gets added to a priority queue on the ledger. The priority queue operates based on priority and time to differentiate work value and prevent starvation for lower priority work requests. Workers should also choose work based on their resources and capabilities, but we leave the optimal dequeuing of work to future research.

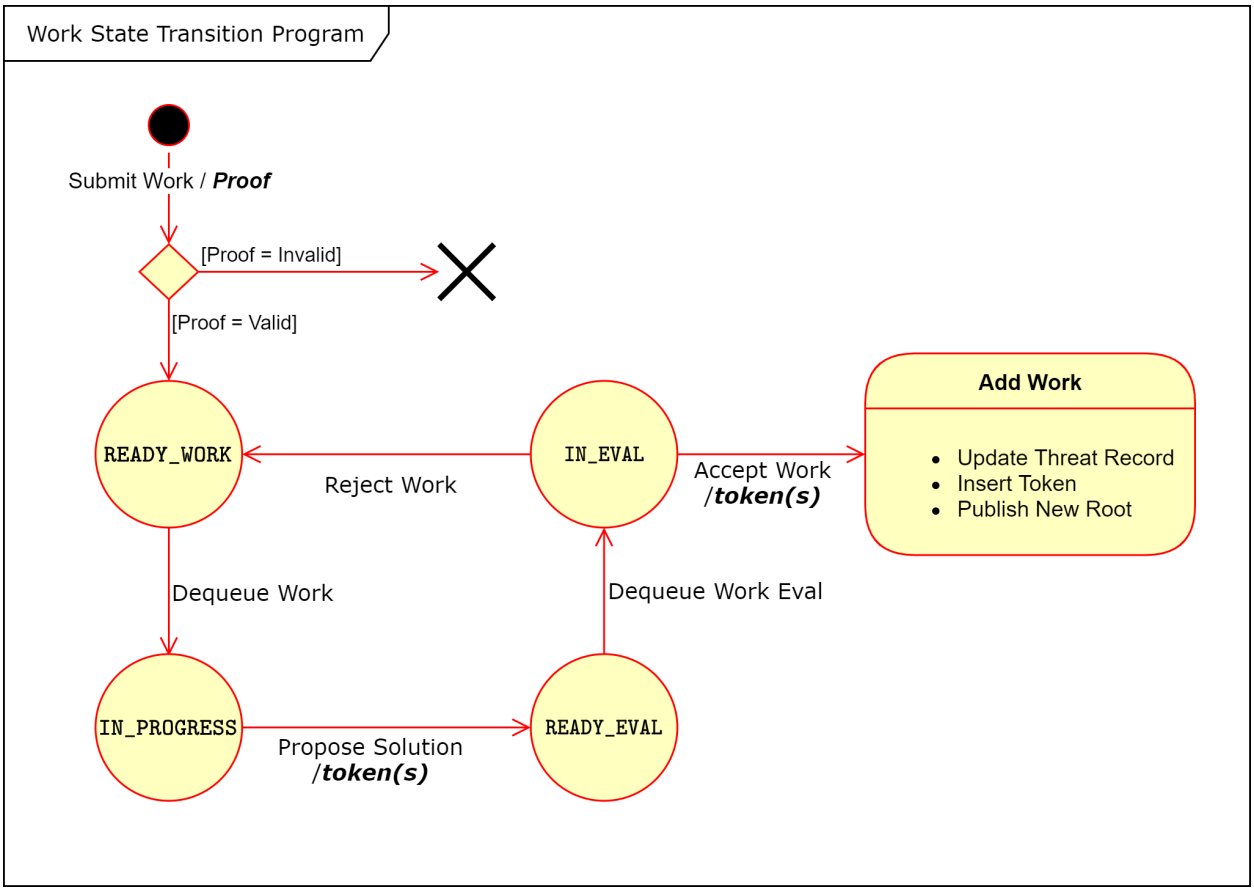

Figure 15: Work State Transition.

Finally, the ledger adds an evaluated solution by i) updating the `event_record` with the added context provided through the work solution, ii) inserting the tokens provided with the work solution and evaluation, and iii) publishing a new root to the network based on the updated tokens.

The entity requesting work will likely search for the work solution periodically. Thus, the network supports authentication-only proofs using tokens to preserve the anonymity of the work requester. An entity need not authenticate with an identity, save only to perform work.

4.6 Implementation

We performed testing to evaluate the Merkle tree maintenance from section 4.4.2 and token authentication in section 4.4. Also, we propose implementation guidelines for implementing the blockchain under realistic loading conditions. Our tests of the zk-SNARK proofs use snarkjs and

circomlib, and the performance was tested on an Intel Core i5-8356U CPU @1.60 GHz with 16GB of RAM.

4.6.1 Token Authentication Performance

The Merkle tree height drives the network performance for token authentication in both storage and time. Authentication allows sparse tree storage at both the organizational, h_{org}, and user levels, h_{user}. However, the network must provide frequent updates at the network level, h_{net}, to support anonymous authentication. Due to the frequency of these updates, we propose setting h_{org} at 15, which for a 256-bit node size, requires 1 MiB of storage.

Users can manage a much deeper portion of the Merkle tree because they only store the sparse tree based on the number of tokens they possess, but the token proof circuit requires a consistent depth. Figure 16 shows the relationship between the depth and proof times. Here, we propose a reasonable tree depth of, at most, 128, which provides ample space for both the foreseeable maximum number of organizational users and the number of tokens allocated for each user.

The parameters and performance of the algebraic circuit for a tree with this size are shown in Table 7.

4.6.2 Ledger Operation Guidelines

We developed the chaincode model for use in Hyperledger Fabric, and although a full-scale simulation is in development, we make some observations here about the operation of the network.

There are three types of transactions proposed: i) event reports, ii) work management, and iii) network maintenance activities, including Merkle root updates. To develop a realistic expectation of throughput, we consider the critical infrastructure sectors in the United States. The Department of Homeland Security identifies sixteen critical infrastructure sectors [28]. Using utility data from the Energy Information Administration [6], we find 3,338 individual utility companies in the electric sector. If each organization produced an average of ten transactions per day during a peak of four working hours, we could expect a maximum throughput of 40 transactions per second.

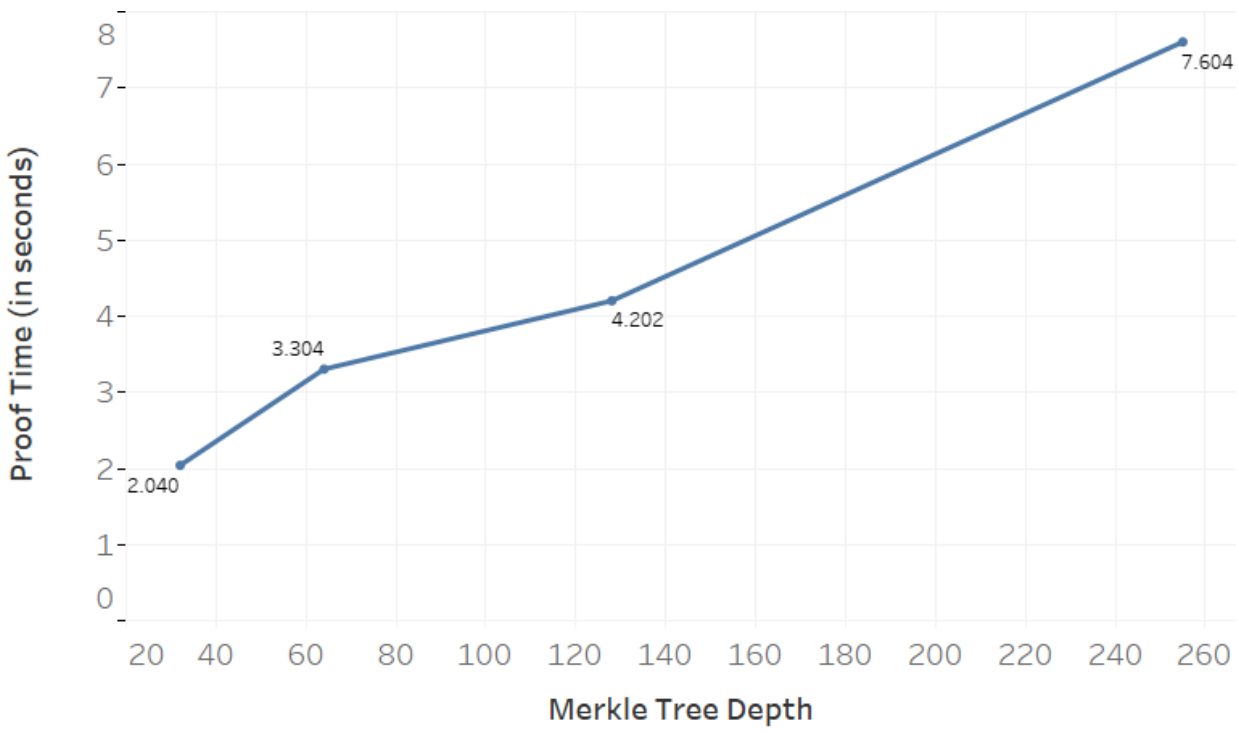

Figure 16: Proof Times Relative to Merkle Tree Height.

Table 7: Sparse Merkle Tree Proof Circuit Parameters and Performance

Merkle Tree Height	128
Number of Wires:	32,486
Number of Constraints:	32,363
Private Inputs:	130
Public Inputs:	2
Number of Labels:	151,304
Number of Outputs:	0
Proof Time:	4,200 ms
Verification Time:	28.5 ms

For this level of throughput, Hyperledger Fabric benchmark experiments indicate a latency of approximately 1 second with a block size of 10 transactions [82]. They also indicate that an endorsement policy of up to four network peers for each transaction would have a minimal effect on the overall transaction latency. Overall, the system's theoretical bounds would fall well within the efficient operating conditions of Hyperledger Fabric.

4.7 Conclusion

We propose a new approach for overcoming the trust barriers of inter-organizational threat intelligence sharing using a distributed ledger technology. We have demonstrated a novel use of zk-SNARKs and Sparse Merkle Trees to enable anonymous authentication and anonymous token spending for the ledger's permissioned users. The results pave the way for a new approach to cybersecurity threat intelligence sharing, which commoditizes the work of CTI curation and sharing to produce a greater cooperative value.

References

[1] *A Common Cyber Threat Framework: A Foundation for Communication*. 2013.

[5] Elli Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the Thirteenth EuroSys Conference*. 2018, pp. 1–15.

[6] *Annual Electric Power Industry Report*. https://www.eia.gov/electricity/data/eia861/. Accessed: 2021-03-12.

[9] Aritra Banerjee, Michael Clear, and Hitesh Tewari. "Demystifying the Role of zk-SNARKs in Zcash". In: *2020 IEEE Conference on Application, Information and Network Security (AINS)*. IEEE. 2020, pp. 12–19.

[10] Sean Barnum. "Information with the Structured Threat Information eXpression (STIX™)". In: (2013).

[11] Nir Bitansky et al. "Succinct non-interactive arguments via linear interactive proofs". In: *Theory of Cryptography Conference*. Springer. 2013, pp. 315–333.

[13] Sean Bowe, Ariel Gabizon, and Matthew D Green. "A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2018, pp. 64–77.

[15] Ernie Brickell, Jan Camenisch, and Liqun Chen. “Direct anonymous attestation”. In: *Proceedings of the 11th ACM conference on Computer and communications security*. 2004, pp. 132–145.

[17] J. Camenisch et al. “One TPM to Bind Them All: Fixing TPM 2.0 for Provably Secure Anonymous Attestation”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 901–920.

[18] Jan Camenisch, Manu Drijvers, and Anja Lehmann. “Anonymous attestation using the strong diffie hellman assumption revisited”. In: *International Conference on Trust and Trustworthy Computing*. Springer. 2016, pp. 1–20.

[19] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. “Compact E-Cash”. In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 302–321.

[20] David Chaum. “Security without identification: Transaction systems to make big brother obsolete”. In: *Communications of the ACM* 28.10 (1985), pp. 1030–1044.

[28] *Critical Infrastructure Sectors*. https://www.cisa.gov/critical-infrastructure-sectors. Accessed: 2021-03-12.

[29] *Cyber Risk Information Sharing Program (CRISP)*. Tech. rep. Accessed: 2021-01-13. Department of Energy, Office of Cybersecurity, Energy Security, and Emergency Response.

[30] George Danezis et al. “Pinocchio coin: building zerocoin from a succinct pairing-based proof system”. In: *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*. 2013, pp. 27–30.

[31] Michael Daniel and Joshua Kenway. “Repairing the Foundation: How Cyber Threat Information Sharing Can Live Up to its Promise and Implications for NATO”. In: *Cyber Threats and NATO 2030: Horizon Scanning and Analysis* (), p. 178.

[32] Constance Douris. *Cyber threat data sharing needs refinement*. Lexington Institute Arlington, Virginia, 2017.

[34] *Enhanced Cybersecurity Services (ECS)*. Tech. rep. Accessed: 2021-01-13. Department of Homeland Security.

[35] *Exploring the opportunities and limitations of current Threat Intelligence Platforms*. Tech. rep. Accessed: 2021-01-06. ENISA, Dec. 2017.

[41] “Global Security Operations Center Market Forecast up to 2025”. In: *Business Wire (English)* (2019).

[42] Seonghyeon Gong and Changhoon Lee. “Blocis: blockchain-based cyber threat intelligence sharing framework for sybil-resistance”. In: *Electronics* 9.3 (2020), p. 521.

[43] *Hackerone List of Bug Bounty Programs*. https://hackerone.com/bug-bounty-programs. Accessed: 2021-03-11.

[44] Shen He et al. "BloTISRT: Blockchain-based Threat Intelligence Sharing and Rating Technology". In: *Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies*. 2020, pp. 524–534.

[48] Office of Inspector General. *DHS Made Limited Progress to Improve Information Sharing under the Cybersecurity Act in Calendar Years 2017 and 2018*. 2020.

[49] Christopher Johnson et al. *Guide to cyber threat information sharing*. Tech. rep. National Institute of Standards and Technology, 2016.

[51] Ahmed Kosba et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 839–858.

[57] Anna Lysyanskaya et al. "Pseudonym systems". In: *International Workshop on Selected Areas in Cryptography*. Springer. 1999, pp. 184–199.

[61] Ian Miers et al. "Zerocoin: Anonymous distributed e-cash from bitcoin". In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 397–411.

[63] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. Manubot, 2019.

[69] Maksym Petkus. "Why and How zk-SNARK Works: Definitive Explanation". In: ().

[72] Raúl Riesco, Xavier Larriva-Novo, and Vıctor A Villagrá. "Cybersecurity threat intelligence knowledge exchange based on blockchain". In: *Telecommunication Systems* 73.2 (2020), pp. 259–288.

[75] Eli Ben Sasson et al. "Zerocash: Decentralized anonymous payments from bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. IEEE. 2014, pp. 459–474.

[77] Meng Shen et al. "Blockchain-based incentives for secure and collaborative data sharing in multiple clouds". In: *IEEE Journal on Selected Areas in Communications* 38.6 (2020), pp. 1229–1241.

[80] Ryan Stillions. *The DML Model*. 2014.

[82] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. "Performance benchmarking and optimizing hyperledger fabric blockchain platform". In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE. 2018, pp. 264–276.

[83] *The Value of Threat Intelligence: A Study of North American and United Kingdom Companies*. Tech. rep. Accessed: 2021-01-06. Ponemon Institute, July 2016.

[85] Cynthia Wagner et al. “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*. 2016, pp. 49–56.

[89] Gavin Wood et al. “Ethereum: A secure decentralised generalised transaction ledger”. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[93] Kim Zetter. “Exclusive: Comedy of Errors Led to False ‘Water-Pump Hack’Report”. In: *Wired Threat Level* (2011).

[97] Adam Zibak and Andrew Simpson. “Cyber threat information sharing: Perceived benefits and barriers”. In: *Proceedings of the 14th international conference on availability, reliability and security*. 2019, pp. 1–9.

5 Future Work

The results in chapter 2 make several assumptions about the adversary and their placement in the network model. An entity will likely have more threat information about the adversary, such as their capabilities, intent, and motivation. The automated analysis could better approximate reachability by more accurately modeling the adversary's interaction with the vulnerability.

Also, the named entity recognition over vulnerability descriptions can systematically extract additional features to assist in the situation understanding of the vulnerability. The NLP model we have developed works well to extract network service features. However, more annotation work and word vector embeddings are necessary to use the model more generally in understanding vulnerabilities.

In chapter 3, the word vector similarity metrics indicate that our corpus vocabulary does not represent many software vendor and product words. We were able to fabricate the word vectors through representation in synthetic texts, but this approach increases similarity for all software vendors and products. More generally, we need a better approach for producing synthetic text for out-of-vocabulary words. Higher quality vulnerability word vectors should also improve the efficiency in annotation for the named entity recognition problem.

The CPE recommender solution will likely support matching natural language across multiple vulnerability datasets. One such application includes Bug Bounty Programs (BBP). BBP provides a crowdsourcing approach for software publishers to find and fix security vulnerabilities missed in software development. A bounty compensates individuals for publicly disclosing vulnerabilities in hopes of increasing the number of people reviewing the software for weaknesses, or in the worst case, provides an incentive to publicly report the vulnerability instead of selling to an adversary [3]. Bounty hunters have several options, from large tech firms to platforms such as HackerOne and Bugcrowd. The platforms comprise hundreds of software publishers advertising bounties for specific types of vulnerabilities. Through platform services, independent bounty hunters connect with the software publishers offering the bounty. Analysis of the rules of engagement performed in [54] shows the typical program specifications of eligible and non-eligible vulnerabilities, pro-

hibited actions, and reward evaluations. Individuals may submit vulnerabilities for review, and bounties are publicly accepted and paid out.

With automated CPE identification from natural language, we can better study BBP and the NVD correlation. For example, we can study the causation relationship between vulnerability disclosure and working exploit code. Additional natural language about vulnerabilities exists in BBP, expanding data features to improve learning models and recommenders.

Chapter 4 exposes many additional research questions. Although we understand current approaches for exchanging CTI, we do not yet know how private entities and intelligence agencies would adapt to more actionable and current intelligence sharing if our approach were adopted. The economy of threat sharing for cooperative benefit also requires more research to understand the value gained and the necessary reward to incentivize sharing. Finally, more open CTI curation provides opportunities to study automation in machine learning and natural language processing over event reports.

6 Overall Conclusions

This dissertation has presented improvements to cyber situational understanding in both cybersecurity vulnerability and threat analysis. These two analytic components constitute the external influence on cybersecurity risk to an entity and feed into the command and control decisions to better defend against adversarial attacks.

In chapter 2, we present an approach to assessing vulnerability exposures by automating both (i) the adversarial reachability of each vulnerability and (ii) the safe-state analysis of each reachable vulnerability. To do so, we overcame the challenges of modeling adversarial movement in a network and extracting key network service features to associate public vulnerability datasets to entity-defined network security policies. As a result, an entity can automatically determine unsafe vulnerabilities in its network. We show that for a common control system network in the electric sector, the automation produces significantly fewer vulnerabilities requiring immediate attention.

Continuing with vulnerability analysis, chapter 3 describes a recommender system to better identify structured CPE URIs from collected hardware and software artifacts in an entity's inventory system. The recommender system uses a combination of fuzzy matching, word vector similarity testing, and machine learning to identify an ordered set of optimal naming URIs for associating vulnerabilities to assets. In our experiments, the resulting system markedly reduces the amount of time required by an analyst to map their inventory and improves the accuracy of the vulnerability source mapping.

Finally, chapter 4 targets the problem of cyber threat intelligence exchange. Using zk-SNARKs, a permissioned blockchain for cyber threat intelligence can be shared and used without entity attribution. This work lays out a new approach to managing Merkle trees associated with zk-SNARKS to support (i) efficient token exchange, (ii) network broadcast, (iii) revocation, and (iv) anonymous authentication. Furthermore, the chaincode used for sharing CTI supports incentives and validation services. A community of trusted public and private entities can use the chaincode to produce less voluminous and more actionable CTI for cooperative benefit.