



Towards Automated Assessment of Vulnerability Exposures in Security Operations

Philip Huff^(✉) and Qinghua Li

University of Arkansas, Fayetteville, AR 72701, USA
{phu, qinghual}@uark.edu

Abstract. 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 paper 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.

Keywords: Software vulnerability · Risk analysis · Artificial intelligence

1 Introduction

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 [5]. 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 [13]. 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.

Our 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.
- 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.
- Evaluation of the solution based on vulnerabilities from the NVD and identification of gateway vulnerabilities that can flip the network attack modeling in favor of the adversary.

Section 2 reviews related work. Section 3 introduces data modeling. Section 4 describes how to identify the network service associated with vulnerabilities. Section 5 presents network reachability analysis under firewall rules. Section 6 presents the safety model for vulnerability exposure checking. The last two sections present evaluation results and conclusions.

2 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 [28], 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 [24]. 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 [19]. Recent solutions in automated AD-Tree generation [12], multi-parameter risk optimization [15] and automatically relating attacks to attack tree goals [23] 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 [27] and [11]. In [33], 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 [31] 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 [17, 34], they propose metrics for a qualitative security score based on vulnerabilities present in the network. Similarly, [26] 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 [16].

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 [20] and, more recently, in trusted computing [35]. 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. [21] 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. [32] uses ML models for attack classification and improved impact scoring, and [22] uses concept drift in NLP to assess vulnerabilities based on their descriptions. [36], and [37] use machine learning to recommend remediation actions for and predict the probabilistic risk levels of vulnerabilities, and [25] uses natural language processing over vulnerability descriptions to identify mitigation information. [18] studies how to map software assets to vulnerabilities. See [9] 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.

3 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.

3.1 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) [3]. 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 Sect. 6 to precisely define capability in terms of safety.

Table 1. CVSS-based data features

CVSS feature	<i>Vulnerability</i>	<i>Adversary</i>	<i>Asset</i>
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 $\{N, L, H\}$.

3.2 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 Sect. 7, 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.

3.3 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 Sect. 4, and how to explore an adversary’s network reachability to the target device and service in Sect. 5.

4 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.

4.1 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) [1], (ii) Common Vulnerability Scoring System (CVSS) [2] features, and (iii) the Common Weakness Enumeration (CWE) [4]. All of these features are regularly updated and made available by the NVD [6].

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. However, 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. As shown in Table 2, the prediction is very accurate.

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

4.2 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 [8, 14]. We use NER to extract standard features in vulnerability descriptions.

Inspired by existing work on cybersecurity ontologies [10, 29, 30], 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 category	NER results summary		
	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>
SERVICE	76%	68%	72%
SOFTWARE	63%	64%	63%
THREAT	72%	61%	66%
WEAKNESS	70%	54%	61%

4.3 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 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 Fig. 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

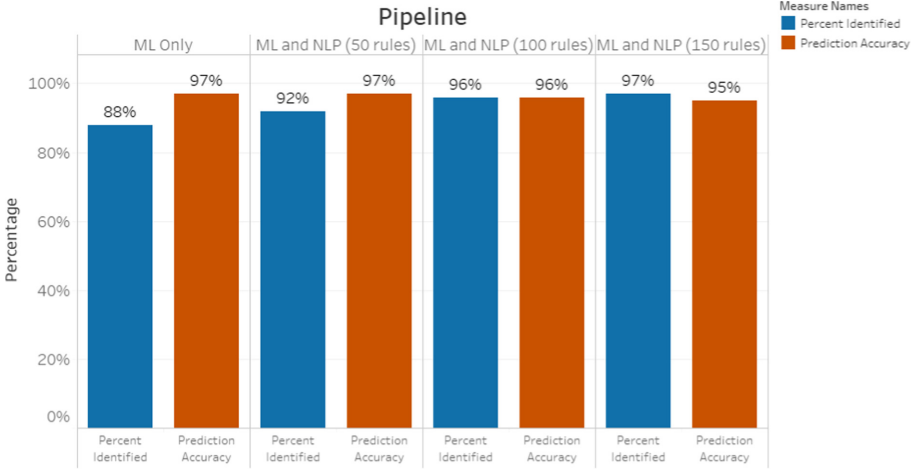


Fig. 1. Performance of network service extraction

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.

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

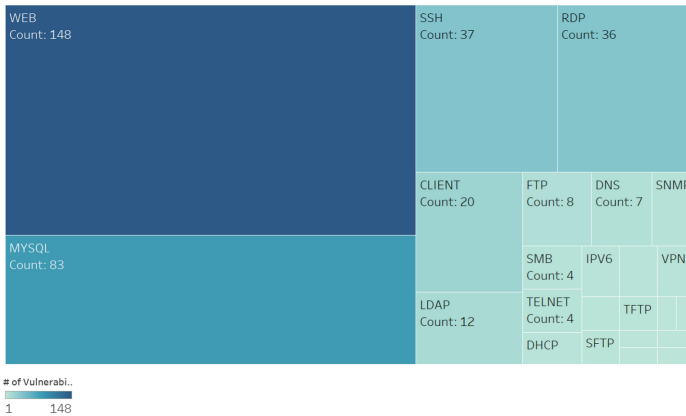


Fig. 2. Additional network services identified through NLP

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.

5 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 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 Fig. 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.

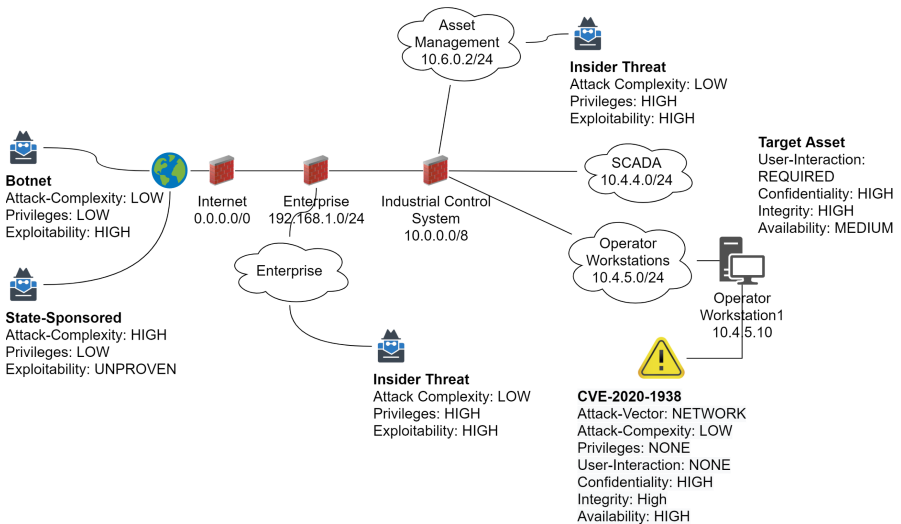


Fig. 3. Reachability analysis combining Adversaries, Targets, and Vulnerabilities

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-View¹) to parse out network topology and accessibility 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 Fig. 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.

6 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 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

¹ <https://www.network-perception.com/>.

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 Sect. 3.1) of *medium* would apply.

Dominance Labels				Impact Gradient Labels			
Attack Complexity (AC)	Privilege (PR)	User Interaction (UI)	Exploit Code Maturity (E)	Confidentiality (C)	Integrity (I)	Availability (A)	
 Vulnerability	High	None	Required	High	Medium	High	Medium
 Target Machine		Required		High	High	Low	None
 Adversary	Low	High		High			
Result	$v > \epsilon$ <i>Vulnerability</i>	$\epsilon > v$ <i>Adversary</i>	$\tau > v$ <i>Target</i>	$\epsilon > v$ <i>Adversary</i>	Medium	Low	None

The dominating vulnerability here indicates a safe state
Overall impact would be *Medium* if the state were not safe

Fig. 4. Example final state labeling

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

- 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:

$$\begin{aligned} (v_s)dom(\epsilon_s, \tau_s) &\iff \\ \exists c \in Cap_\epsilon, v_s^c &> \epsilon_s^c \\ \forall \exists c \in Cap_\tau, v_s^c &> \tau_s^c \end{aligned}$$

And dominance for the adversary and target is defined as:

$$\begin{aligned} (\epsilon_s, \tau_s)dom(v_s) &\iff \\ \forall c \in Cap_\epsilon, \epsilon_s^c &> v_s^c \\ \wedge \forall c \in Cap_\tau, \tau_s^c &> v_s^c \end{aligned}$$

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 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 .

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

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:

$$\exists c \in Cap \mid v^c \cap \epsilon^c \in \emptyset \vee v^c \cap \tau^c \in \emptyset \quad (1)$$

Therefore, the adversary cannot exploit the vulnerability on the target in at least one category, and by Definition 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.

6.2 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:

$$\forall g \in G \mid 0 \leq i < |g|, g_i \subseteq g_{i+1} \quad (2)$$

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

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

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

7 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 [7]. 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 Fig. 5. The data set also includes 4,894 combined software assets mapped to the NVD.

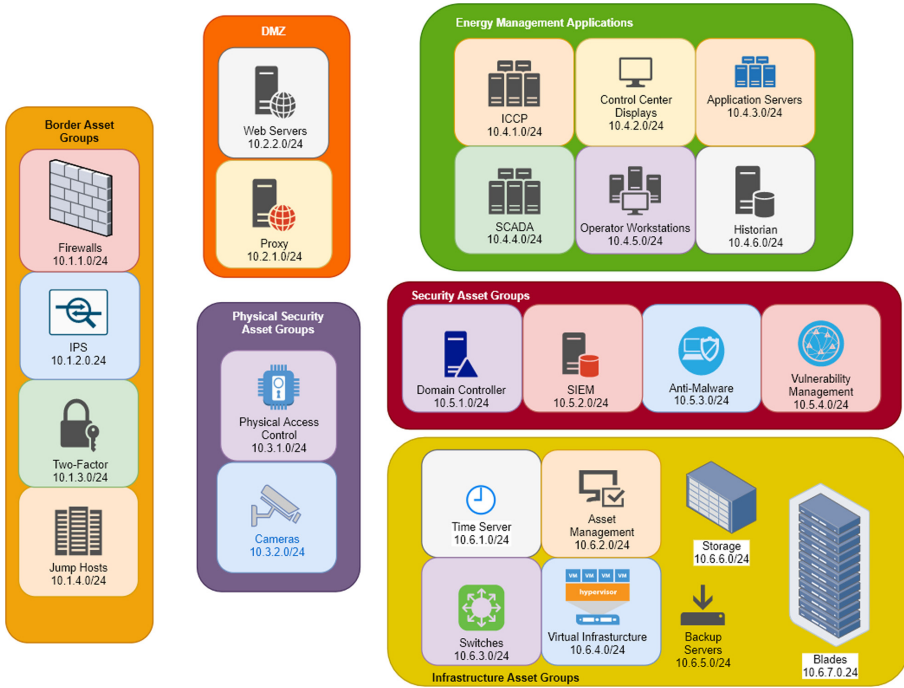


Fig. 5. Sample system network zonal diagram

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 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.

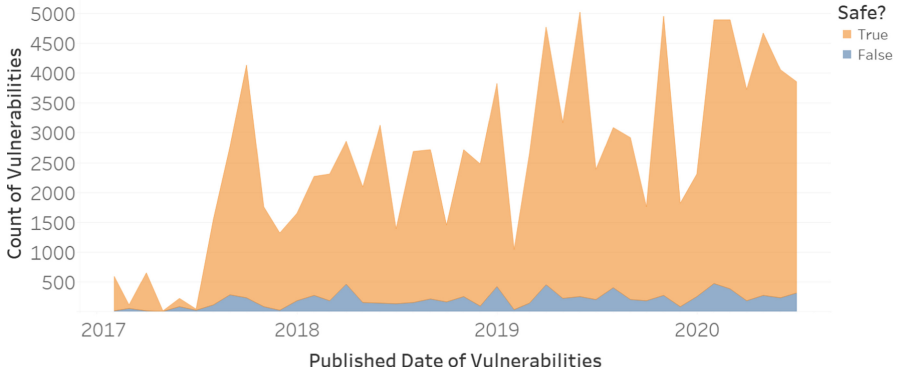


Fig. 6. Monthly safety analysis of all applicable vulnerabilities

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 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 Fig. 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 vulnerabilities beyond the third iteration. The number of unsafe vulnerabilities rises from

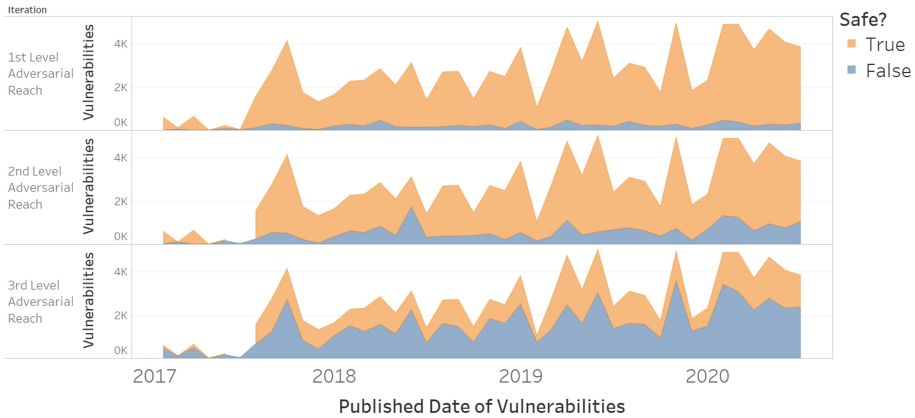


Fig. 7. Iterative safety analysis for all applicable vulnerabilities

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.

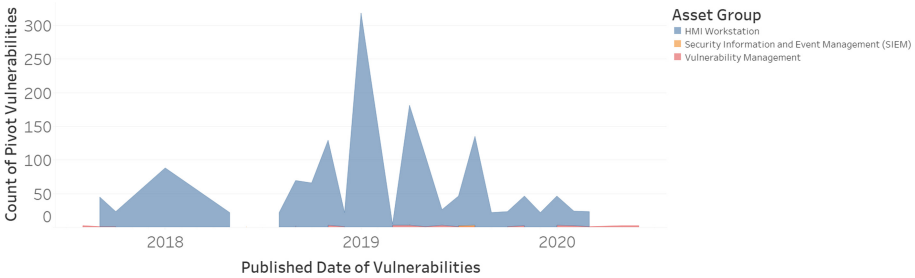


Fig. 8. Vulnerabilities allowing extension of adversarial reach

8 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. 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 apply limited resources better.

Acknowledgement. This material is based upon work supported by the Department of Energy under Award Number DE-CR0000003.

References

1. Common product enumeration standard. <https://nvd.nist.gov/products/cpe>. Accessed 28 Jan 2020
2. Common vulnerability scoring system specification. <https://www.first.org/cvss/v3.1/specification-document>. Accessed 28 Jan 2020
3. Common vulnerability scoring system v3.1: Specification document. <https://www.first.org/cvss/v3.1/specification-document>. Accessed 1 Feb 2020
4. Common weakness enumeration. <https://cwe.mitre.org/>. Accessed 28 Jan 2020
5. National vulnerability database data feed. <https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time>. Accessed 1 Feb 2020
6. National vulnerability database data feed. <https://nvd.nist.gov/vuln/data-feeds>. Accessed 28 Jan 2020
7. North American electric reliability corporation (NERC) critical infrastructure protection (CIP) standards. <https://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx>. Accessed 1 Feb 2020
8. Spacy and prodigy network language processing tools. <https://explosion.ai/>. Accessed 2 Feb 2020
9. Vulnerability and patch management resources. <http://cybersecurity.ddns.uark.edu/vpm/>. Accessed 25 June 2021
10. Stix version 2.1, March 2020. <https://docs.oasis-open.org/cti/stix/v2.1/cs01/stix-v2.1-cs01.html>. Accessed 9 Mar 2021
11. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: ACM Conference on Computer and Communications Security, pp. 217–224 (2002)
12. Audinot, M., Pinchinat, S., Kordy, B.: Guided design of attack trees: a system-based approach. In: 2018 IEEE 31st Computer Security Foundations Symposium (CSF), pp. 61–75, July 2018
13. Collins, K.: The hackers who broke into Equifax exploited a flaw in open-source server software. Quartz. <https://qz.com/1073221/the-hackers-who-broke-into-equifax-exploited-a-nine-year-old-security-flaw/>
14. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* **12**(ARTICLE), 2493–2537 (2011)
15. Fila, B., Wide, W.: Efficient attack-defense tree analysis using pareto attribute domains. In: 2019 IEEE 32nd Computer Security Foundations Symposium (CSF), June 2019
16. Gamarra, M., Shetty, S., Nicol, D.M., Gonzalez, O., Kamhoua, C.A., Njilla, L.: Analysis of stepping stone attacks in dynamic vulnerability graphs, pp. 1–7, May 2018
17. Ghosh, N., Ghosh, S.K.: An approach for security assessment of network configurations using attack graph. In: International Conference on Networks & Communications, pp. 283–288 (2010)

18. Huff, P., Li, Q.: A recommender system for tracking vulnerabilities. In: International Workshop on Next Generation Security Operations Centers (NG-SOC) (2021)
19. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 80–95. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19751-2_6
20. Landwehr, C.E.: Formal models for computer security. ACM Comput. Surv. (CSUR) **13**(3), 247–278 (1981)
21. Le, H.T., Loh, P.K.K.: 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, March 2011
22. Le, T.H.M., Sabir, B., Babar, M.A.: Automated software vulnerability assessment with concept drift. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp. 371–382 (2019)
23. Mantel, H., Probst, C.W.: On the meaning and purpose of attack trees. In: 2019 IEEE 32nd Computer Security Foundations Symposium (CSF), pp. 184–18415, June 2019
24. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006). https://doi.org/10.1007/11734727_17
25. McClanahan, K., Li, Q.: Automatically locating mitigation information for security vulnerabilities. In: IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm) (2020)
26. Noel, S., Jajodia, S.: Metrics suite for network attack graph analytics. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference, pp. 5–8 (2014)
27. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 Workshop on New security paradigms, pp. 71–79 (1998)
28. Schneier, B.: Attack trees. Dr. Dobb’s J. **24**(12), 21–29 (1999)
29. Sikos, L.F.: OWL ontologies in cybersecurity: conceptual modeling of cyber-knowledge. In: Sikos, L.F. (ed.) AI in Cybersecurity. ISRL, vol. 151, pp. 1–17. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-98842-9_1
30. Syed, Z., Padia, A., Finin, T., Mathews, L., Joshi, A.: UCO: a unified cybersecurity ontology. In: UMBC Student Collection (2016)
31. Wang, L., Jajodia, S., Singhal, A., Noel, S.: k -zero day safety: measuring the security risk of networks against unknown attacks. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 573–587. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15497-3_35
32. Wang, P., Zhou, Y., Sun, B., Zhang, W.: Intelligent prediction of vulnerability severity level based on text mining and XGBboost. In: 2019 Eleventh International Conference on Advanced Computational Intelligence (ICACI), pp. 72–77 (2019)
33. Wing, J.M., et al.: Scenario graphs applied to network security. In: Information Assurance: Survivability and Security in Networked Systems, pp. 247–277 (2008)
34. Xie, A., Wen, W., Zhang, L., Hu, J., Chen, Z.: Applying attack graphs to network security metric. In: Proceedings of the 2009 International Conference on Multimedia Information Networking and Security, vol. 01, pp. 427–431 (2009)
35. Xu, M., et al.: Dominance as a new trusted computing primitive for the internet of things. In: 2019 IEEE Symposium on Security and Privacy (SP) (2019)

36. Zhang, F., Huff, P., McClanahan, K., Li, Q.: A machine learning-based approach for automated vulnerability remediation analysis. In: IEEE Conference on Communications and Network Security (CNS) (2020)
37. Zhang, F., Li, Q.: Dynamic risk-aware patch scheduling. In: IEEE Conference on Communications and Network Security (CNS) (2020)