EAI Endorsed Transactions
on Security and Safety
Research Article EAI.EU

Towards Automating the Assessment of Software Vulnerability Risk

Philip Dale Huff[1], Qinghua Li[2]

[1]University of Arkansas, Department of Computer Science and Computer Engineering, pdhuff@ualr.edu
[2]University of Arkansas, qinghual@uark.edu

Abstract

Remediating known software vulnerabilities is one of the most pressing operational challenges for personnel tasked with maintaining a secure system, mainly due to the large amount of vulnerabilities to analyze and mitigate. For computing environments such as data centers, Internet-of-Things infrastructure, and industrial control systems, vulnerability mitigation comes at an even higher cost because of the constant testing, coordination, and configuration change required. We address this problem by identifying a minimal set (i.e., as few as 4%) of the total applicable vulnerabilities with high risks for security personnel to concentrate mitigation effort on. To do so, we introduce a scheme to contextualize vulnerability risk by calculating the adversarial capability of exploiting a vulnerability on a target device. We use a machine learning and natural language processing pipeline to process online vulnerability descriptions and extract network service features, which can then correlate to the system's network firewall policies. The pipeline then allows automated model-checking to assess an unsafe state for a given vulnerability, adversary, and target device.

Keywords: security, vulnerability, risk, artificial intelligence

Received on 10 November 2020, accepted on 15 December 2020, published on 16 December 2020

doi: 10.4108/eai.25-6-2021.170247

1. Introduction

The sheer volume of work required of organizations to protect information systems continuously often exceeds the human workforce capacity. The human workproblem has become increasingly evident in the sphere of vulnerability and patchmanagement, where the number of vulnerabilities reported annually by the National Vulnerability Database (NVD) doubled in 2017 to over 14,000 and continues to rise [7]. With this trend, organizations cross a vexing threshold inwhich the rate of incoming vulnerabilities exceeds their capacity to assess vulnerabilities' risks. As a result, organizations often either (i) blindly patch every vulnerability without analyzing risks, which leads to unnecessary reductions in reliable operation since patching causes system rebooting/downtime, or (ii) fall hopelessly behind if they analyze risks, which increases the duration of time systems remain vulnerable. Whereas numerous automation tools exist for scanning and patching vulnerabilities, few mechanisms exist for providing a deep contextual understanding of how the vulnerabilities influence organizational risk in the computing environment. These conditions have the potential to create a hopelessly flawed environment that leaves effective mitigation of vulnerabilities to chance. The risk management problem with software vulnerabilities goes well beyond the automatic update cycles with which most people have familiarity. Interactive consumer device vulnerabilities are both numerous and sometimes dangerous, but otherwise manageable through regular and automatic update cycles. How- ever, environments for server computing, industrial control systems (ICS), and Internet of Things (IoT) devices present far more exigent challenges in vulnerability mitigation resource planning. For these environments, attacks requiring human interaction (e.g., phishing, drive-by downloads) are a minor concern or altogether irrelevant. Instead, the ability of an adversary to exploit a vulnerability remotely takes precedent. We

find only a small percentage of vulnerabilities pose such imminent risk of a security breach, but the analysis to identify this remnant requires a deep understanding of the vulnerability and the operating environment. For example, in a 2017 Equifax breach, a months-old unpatched Apache Struts vulnerability was identified as the initial attack vector [17]. If the degree of risk were known, one would expect the organization to patch quicker, but this level of analysis often requires manual human labor far beyond the capacity of most organizations.

Our work aims towards automatically identifying the small percentage of software vulnerabilities presenting an immanent risk to an entity. Using an artificial intelligence pipeline complemented with minimal human analysis when necessary, we assess vulnerabilities against a firewall protection policy. To illustrate, consider a new remote code execution vulnerability targeting TCP port445 (i.e., SMB). In our scheme, the network service is immediately extracted and analyzed against a firewall policy. A security analyst can then react to affected systems not protected by the firewall.

The automation is realized through the NVD as one of the most extensive, consistent, and continuously updated open datasets in cybersecurity. The NVD has over 130,000 vulnerability records with a rate of 50 additions per day. Be- cause of the high quality and consistency of this dataset, machine learning, and natural language processing (NLP) models show high potential for work automation. Employing this automation makes possible the extraction of additional features suitable for more contextual and deterministic state-based analysis. In our approach, we extract network service features from the NVD to link vulnerabilities with a physical network topology.

Using the target system's network firewall policy as input to the model, we better understand the true adversarial capability to exploit a given vulnerability remotely. Firewalls represent the front line and primary defense against the active exploitation of software vulnerabilities in the server computing environment. Additionally, because firewalls must incorporate into any network configuration, their configuration is homogeneous across multiple vendors and can be automatically parsed for input. For our research, we use Network Perception's NP Live product to provide this input [10]. Vulnerability, adversarial threat, and asset data features are combined intoa formal model to automate the assessment of a final state safety invariant.Our results show the feasibility of assessing a large number of vulnerabilities applicable to target systems in an automated and contextual manner, thereby drastically reducing workforce requirements and enabling a focus on mitigation of the small number of unsafe software vulnerabilities.

Contributions. Our contributions to automated software vulnerability riskanalysis include the following:

- A formal definition of safety when combining vulnerability, adversarial threat, and target asset features;
- A formal model for tying firewall configuration data to applicable vulnerabilities in an organizational network;
- An artificial intelligence pipeline to extract network service features necessary for automating the association between firewall configuration data and vulnerabilities;
- Demonstrate the potential for analysts to focus mitigation efforts on as few as 4% of the total number of vulnerabilities applicable to their system.

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, originally pioneered by Schneier [29], 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 providesa formal specification for attack trees [26]. 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 [21]. Recent novel solutions in automated AD-Tree generation [16], multi-parameter risk optimization [18] and automatically relating attacks to attack tree goals [25] 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 simpleattacker goal to exploit the system and use standard atomic attributes to measure the attacker 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 at- tacker's capability to exploit vulnerabilities originated in work [28] and [15]. In [32], 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 [30] 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 [33,20], they propose metrics
for a qualitative security score based on vulnerabilities present in the network. Similarly, [27] combines vulnerability metric data with firewall topology to pro-vide 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 [19].
AD-Trees and attack graphs have the same nuisance of saturating the securityanalyst with output data. Our approach overcomes this obstacle by focusingmore narrowly on the common problem of software vulnerability managementusing standard data features (i.e., CVSS) well understood by practitioners. We abstract much of the complexity in decision making using set dominance similarto other areas of formal models in computer security such as access control [22] and, more recently, in IoT trusted computing [34]. Also, our approach improves on the network attack graph model by moving closer to the risk result. Insteadof outputting a graph or risk score which still need much manual analysis to decide whether vulnerabilities need mitigation or not, our model generates a deterministic result as to whether vulnerabilities are safe from attackers or not.Automated feature expansion of the CVSS has been studied in some work.
In [23], they use 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. Their scheme allows generally describing classes of vulnerabilities associated with a given Common Weakness Enumeration (CWE) in terms of language parts. [35] performs both NLP and ML over the NVD features to automate the extraction of security metrics. Recent work in [31] shows high accuracy attack classification and improved impact scoring through ML, and [24] presents the problem of concept-drift in NLP over vulnerability descriptions with high temporal distance. However, these work do not address risk assessment which is the focus of this paper.

3. Dominance Scheme for Software Vulnerabilities

This section presents a definition of safety with respect to a labeling of features associated with software vulnerabilities. Throughout the model discussion, we refer to the transition system in the following definition:
Definition 1. Software Vulnerability Transition System
–$R \subseteq S \times S$ - Transition functions
–$S_0 \subseteq S$ - The set of initial states
–AP - Set of atomic propositions
–$L : S \quad 2^{AP}$ - Labelling of states formula
–Φ = AG(unsafe) - Invariant condition defining an secure state
–S_s - Set of final accepting states
The safety invariant, Φ, is used to indicate an unsafe state cannot be reached by the mode, which in this case means the vulnerability cannot be exploited.
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 on which to assess system safety. Together, these objects provide the propositional labels applying to a system state, such that AP = {v, τ, ϵ}.
Objects have labels associated with the Common Vulnerability Scoring System (CVSS). CVSS is an open standard maintained by a special interest group under the Forum of Incident Response and Security Teams (FIRST) [3] and is broadly used by significant software publishers to describe security vulnerabilities in their software. There are twelve identified CVSS labels applying to the target machine based on the assignment of an associated vulnerability. The National Vulnerability Database assigns the CVSS labels to a centralized and open database of software vulnerabilities.
Labeling vulnerabilities is straight forward, but to define safety in terms of a dominance relation, we must also apply the same vulnerability labels to both target devices and adversaries. We consider each of the CVSS exploitability, temporal and impact metrics in turn as they apply to vulnerabilities, v, adversaries, ϵ, and target devices, τ .
There are two fundamental types of state labels, which we refer to as dominance and impact gradience. With dominance labels, the relationship determines safety between the v, ϵ, and τ , whereas impact gradient determines the degree of impact posed to a target by an exploited vulnerability.
Figure 1 provides an example of how object labels combine in a final state, Ss, 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 3.2) of medium would apply.
We now describe each type of label and how the labeling function applies to the vulnerability in the final state.

3.1 Dominance Labels

First, dominance labels describe vulnerabilities by the capabilities necessary for exploitation, and are de-fined as follows:
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 la- bels, which we formally describe in appendix A

– The label group defines a necessary condition for exploitation.

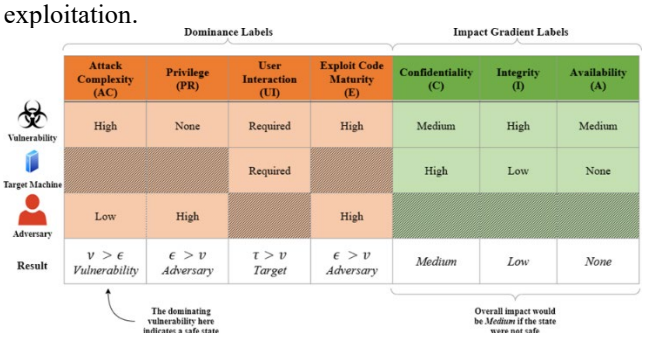

Fig. 1. Example Final State Labeling

This definition holds for the CVSS exploitability and temporal metrics. Labels have order applied as de-scribed 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.

The Attack Vector, AV , label defines the adversarial access necessary to exploit a vulnerability. 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 v and increasing exploit opportunity for the adversary. This label requires reachability analysis of target network described in section 4.1.

Attack Complexity, AC, describes the difficulty required to develop an exploit for a given vulnerability and applies both to objects v and ϵ. The label AC has no meaning for a device, and thus, τ has no value. Propositions include Low, L, and High, H, with the ordered set AC = {L, H}. For example, low attack complexity requirements would indicate a greater opportunity for an adversary 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 exploit opportunity.

The User Interaction label, UI, indicates the degree to which a human must be involved to exploit the vul-nerability. Besides describing v, the label also de- scribes the target device, τ , because certain devices such as servers do not have user interaction. 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 consequently, a server device would most likely have the label N .

The temporal metric of exploitability, EX, also exhibits simple dominance as it applies to both v and ϵ. EX indicates the overall level of exploitability independent of any other characteristic of in v, τ , and ϵ. It de-scribes the cur- rent availability of code to exploit the vulnerability. This label applies to the adversary in terms of capability. A nation-state adversary may be capable of developing exploit code even when the temporal metric considers the exploitability unknown. Likewise, almost any adversary could use publicly available exploit code. 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 and increasing capability of an adversary.

3.2 Impact Gradient Labels

Impact gradient labels apply only to the vulnerability and target device because they describe the impact an exploited vulnerability might have on a 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*}.

3.3 Defining Safety

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

$(v_s)\text{dom}(\epsilon_s, \tau_s) \Longleftrightarrow$
$\exists d \in D_\epsilon, v^d_s > \epsilon^d_s$
$\vee \exists d \in D_\tau, v^d_s > \tau^d_s$

And dominance for the adversary and target is defined as:

$(\epsilon_s, \tau_s)\text{dom}(v_s) \Longleftrightarrow$
$\forall d \in D_\epsilon, \epsilon^d_s > v^d_s$
$\wedge \forall d \in D_\tau, \tau^d_s > v^d_s$

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 captures the definition of safety, which we prove in appendix A.

Theorem 1. if $(v_s)\text{dom}(\epsilon_s, \tau_s)$, then the state, $s \in S_s$ is safe.

3.4 Measuring Impact

Impact gradient labels apply when the final state of a system is not safe, and these labels provide further context for assessing the vulnerable state of a systemand 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:

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

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:*

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

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

3.5 Maintaining Target Device and Adversary Labels

The labeling of state objects necessary to reap the benefits of automation should require minimal effort. Entities such as the NVD already manage and make avail- able vulnerability labels, v in aggregate. However, target device labels, τ, remain the responsibility of the entity managing each system. These labels mostly apply manually but should apply to groups of devices with similar characteristics rather than individual targets.

On the other hand, adversary labels, ϵ, are discretionary and require more consideration. Recall, the adversary labels include $D_\epsilon = \{AC, PR, EX\}$, and these should be selected to match the real adversarial capability closely. We start with assuming a threat actor on the Internet with *High* attack complexity, *Low* privilege and *Unproven* exploitability. The highly resourced threat actor models, for example, a state-sponsored attacker highly capable of exploiting vulnerabilities but may not have privileges on the target.

The Internet adversary works well for vulnerabilities involving client-baseduser interaction, such as through a browser or mobile device. In analyzing 3 yearsof NVD data from 2017 to 2019, 62% of all vulnerabilities fall in this category. For vulnerabilities having an attack vector of *Local*, *Physical* or *Adjacent*, the adversary profile might commonly be labelled *Low* attack complexity, *High* privilege and *High* exploitability, which means the adversary with proximity is highly privileged but may not be as capable in exploiting vulnerabilities.

A scenario may call for assessing multiple adversaries. For example, in the next section when assessing reachability, different types of adversaries could reach the vulnerable target device.

4 Network Reachability

The attack vector label, AV , describes the physical presence needed by the adversary to exploit a vulnerability. Physical vulnerabilities require the adversary to have a hands-on physical presence with the target device. Adjacent means the adversary must be within physical proximity using protocols such as Blue- tooth or WiFi. A Local attack vector requires interactive access on the target operating system, which does not necessarily require physical access. A Network attack vector means the vulnerability is remotely exploitable.

In the 2017-2019 NVD dataset, over 98% of all vulnerabilities have either Local or Network attack vectors. For Local and other non-remote attack vectors, our scheme assumes a capable adversary and assesses safety on other attributes of the system described in section 3. However, for the Network attack vector, the system needs to assess whether an exploitable vulnerability on a given target device is accessible to an adversary over the network.

4.1 Assessing Reachability

Assessing network reachability first requires feature extraction of a given vulnerability to determine the impacted network services. Section 5 describes this process. Then, the model uses firewall configuration analyzers to parse out net- work topology and accessibility between network zones and determine whether a given adversary has an opportunity to exploit a given vulnerability.

The basic case of reachability analysis occurs when an adversary has direct access to exploit the vulnerability through the combined firewall ruleset. How- ever, the state machine also models the stepping-stone case in which an adversary extends its reach into the network through the possession of authentication credentials and vulnerability exploits. To model the adversarial movement, we categorize firewall rules as either interactive or non-interactive services. Inter- active services (e.g., SSH, RDP) allow the adversary authenticated access on the target device, 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.

The reachability model introduces two new definitions to model the adversarial movement towards a target in a computing network.

Definition 4 (Network Zones). Groupings of devices in a computing network to which access control applies.

Z - The set of network zones in a given network topology

$p \in Z$ - Pivot network zones which an adversary may use in moving closer to the target, τ

$\eta \in Z$ - Target networks in which the target, τ resides

For purposes of assessing reachability of an adversary to a given target, we define the following state labelling functions for firewall rules associated with a given network zone in Z:

Definition 5. Firewall Rules

Γ - The set of all network service labels

Ψ ⊆ Γ - Interactive access network services

δ : Z × Γ → Z - The set of firewall rules for a given network zone

The firewall rules, δ assign to network zones in the same way firewalls enforce policy in the network. These include the five components commonly used in firewall rulesets: source & destination IP address, transport service, and source & destination port. Γ refers to the categorized service which we map to target vulnerabilities, and Ψ is a subset of Γ , which we use to define interactive services (e.g., SSH, RDP) permitting adversaries to pivot in networks closer to the target machine.

We concisely model this reachability test through the following grammar. The grammar accepts all possible conditions in which the adversary is able to reach the target machine, and the grammar rejects cases where the combined firewall policy blocks reachability.

⟨Reach⟩ |= ϵ⟨P⟩⟨T⟩

⟨P⟩ |= pΨ ⟨P⟩ | λ

⟨T⟩ |= Ψη | Γητ

The grammar describes reachability with an adversary, ϵ, pivoting to a target. In the second line, pivoting expands to either null (i.e., the adversary can reach the target network directly) or to recursively pivot through interactive service firewall rules. Then, in the last line, the target is expanded to include either interactive access to the target network or non-interactive access to the target device.

Figure 2 shows an example of this grammar, in which the adversary ϵ0 has reachability to the target machine, whereas the aggregate firewall policy blocks the adversary ϵ1.

To implement the grammar, we observe the need to represent the network as both (i) a graph to find pivot networks and (ii) a graph matrix to assess reach- ability. First of all, a single directed acyclical graph (DAG) can represent the pivot graph with edges between network zone vertices and the edges comprised of allowed interactive network services Ψ . The following relation defines a pivot for a directed edge (u, v):

δ(u, Ψ) = v (2)

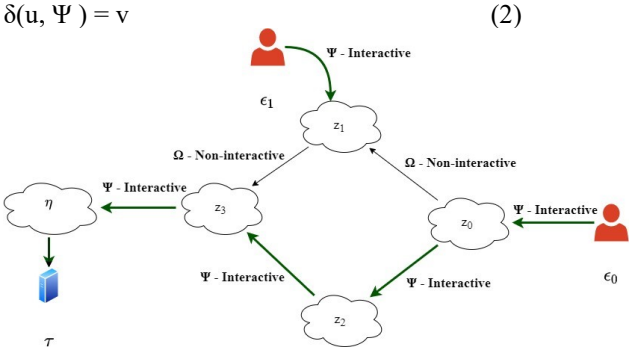

Fig. 2. Assessing Network Reachability

We refer to the states and relations as the *core-graph* needed for modelling pivots. Secondly, the graph matrix, *Reach*, is used for defining the effectivefirewall ruleset between any two network zones:

Reach : S × S → δ (3)

We build Reach by performing a depth-first traversal in the reverse direction of the DAG for each target network, as described in Algorithm 1. This algorithm builds the matrix for all possible connections for efficient reachability assessmentbetween an adversary and a target machine. Line 4 calls the depth-first recursive function for each network zone. Then, the recursive Build-Target-Reach function performs two operations to calculate the combined ruleset between the target network η an adjacent network u.

The first operation in line 9 calculates the intersection of firewall rules be- tween the existing set of firewall rules and the combined ruleset up to the adjacent network zone v. In other words, the adversary in network zone u is restrictednot only by u's firewall ruleset but also by any other firewalls in between the adversary and target machine.

The second operation on line 10 is a less likely scenario in a routed network but accounts for the possibility of having multiple directed paths to the target network. Once we have built the Reach matrix, it can assess reachability betweenany adversary and target machine on the network. Furthermore, Reach provides the network services, which we map directly to the applicable vulnerabilities.

4.2 Reachability Model Analysis

The time complexity for determining reachability between any adversary and any target machine is asymptotically bound by the time required to build the *Reach* matrix. In Algorithm 1, this includes the outer recursion for each network.

Algorithm 1 Building the Reach Matrix

1: **function** BUILD-REACH
2: Initialize the *Reach* matrix and δ rules
3: **for all** $\eta \in z$ **do**
4: BUILD-TARGET-REACH(η, η)
5: **end for**
6: **end function**
7: **function** BUILD-TARGET-REACH(η, v)
8: **for** vertex $u \in z.Adj[v]$ **do**
9: $\delta_{tight} = Reach[\eta][v] \cap \delta(u)$
10: $Reach[\eta][u] = Reach[\eta][u] \cup \delta_{tight}$
11: BUILD-TARGET-REACH(η, u)
12: **end for**
13: **end function**

zone in z and the recursive depth-first traversal to calculate the effective firewall ruleset. The recursive call takes at most:

$$|z| \times \Delta : \Delta = max(\bigcup_{v \in z} |\delta(v)|) \quad (4)$$

Therefore, the worst case time complexity for the *Reach* matrix is:

$$\Theta(|z|^2 \times \Delta) \quad (5)$$

The *core graph* can be formed through *Reach* using a simple breadth-first- search (BFS) of linear time complexity based on the number of network zones. Then, once *Reach* and the *core graph* have formed, the model can assess reachability between any adversary and any target machine in constant time.

Interactive Network Service Labeling -

One of the overall outcomes of our model, which we introduce here, includes the expansion of interactive network services. In a safe state, interactive services include only those allowable through the firewall for direct interaction inside a network zone. Hōwever, as ān output of our model, we identify potentially exploitable non-interactive network-services, which may serve as new network attack-vectors in which the network service becomes a gateway to direct interaction.

A more disruptive change occurs when a network gateway is itself the target device and has an exploitable vulnerability. A gateway has the potential to enable interactive access to all outgoing connected vertices in the DAG.

We model both of these by labelling the affected edges as interactive. If the change requires an update to the Reach matrix, then we must reassess the model for both reachability and safety. It may be possible for several recursions to trigger, but the number of network zones still bounds the time complexity.

5. Artificial Intelligence Pipeline for Feature Expansion

The key enabling feature for automation of the AD tree model is the extraction of network service features from the vulnerabilities. The absence of this extraction creates a burdensome manual task of reviewing every vulnerabilityto identify the impacted network service, which makes the model all but unusable. While the NVD does not describe vulnerable network services as part of the standard reporting mechanism, the available NVD features nicely coalesce to extract network services with high accuracy.

We use a combination of machine learning and natural language processing to optimize the automation of vulnerable network service feature extraction,as shown in Figure 3. The pipeline first uses machine learning classification and then uses natural language processing to extract the network service feature. Two threshold parameters, σ and μ, are defined respectively for machine learning andNLP for determining the degree at which we accept the result. The parameterσ applies to machine learning and includes parameters for both precision and sample support for automatically accepting a result. Likewise, the parameter μ is a running accuracy score of the NLP, which we detail in appendix C. Whenthe accuracy of NLP is not high enough for some vulnerabilities, manual analysis could be used as a complement to identify the network service of these vulnerabilities. In our experiment, less than 0.5% of the vulnerabilities would need manual analysis. Algorithm 2 shows the pseudo-code of this process.

Algorithm 2 Extracting Network Services

```
function EXTRACT-NETWORK-SERVICE(cve)
pred ← net clf.predict(cve.features)
if pred.precision ≥ σ_p & pred.support ≥ σ_s then
return the prediction made by net clf
else
pred_nlp ← net_nlp(cve.description)
if pred_nlp.score ≥ μ then
return pred_nlp
else
    return Manual classification
end if
end if
end function
```

5.1 Machine Learning Classification

In the first level of network service feature extraction, we predict the network- service using NVD features for the training data. In particular, we use the Com-mon Product Enumeration (CPE) [1], Common Vulnerability Scoring System (CVSS) [2] features and the Common Weakness

Enumeration (CWE) [4]. All of

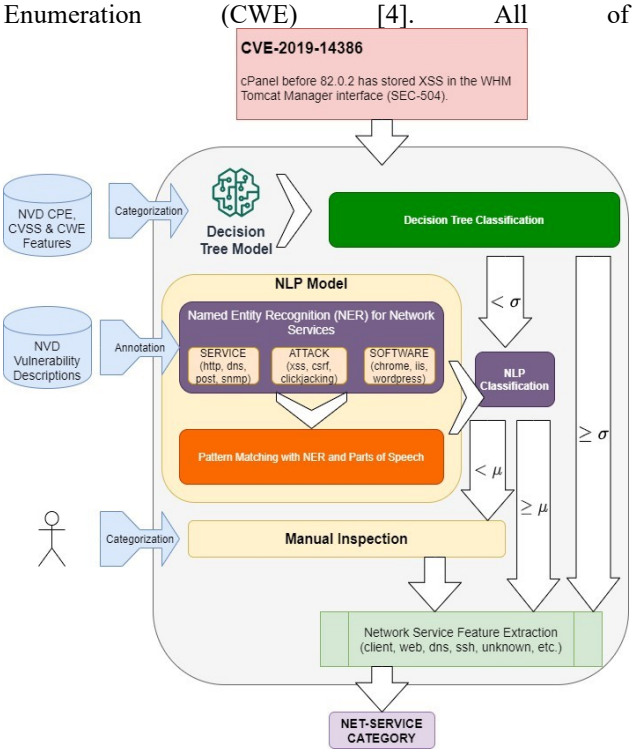

Fig. 3. Network Service Feature Extraction

these features are regularly updated and made available through the NVD [8]. The NVD daily releases vulnerabilities, and tens of thousands of vulnerabilities are categorized each year.

To generate the labeled data set used for training and testing, we labeled the network services for over 19,000 vulnerabilities sampled from the 2017-2019 dataset. Many of the classifications were deterministic through the vulnerability features. For example, the CVSS attack vector (AV) feature has local and physical categories. Because the local and physical features mean the adversary needs direct access to the computer, we classified the network service as client to indicate the exploitation of the vulnerability requires client access to the applicable asset. Our use of client as a label associates with client/server network protocol. The label 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 CWE is a useful feature because it identifies classes of vulnerabilities commonly associated with specific network services. For example, a large number of cross-site scripting and SQL injection vulnerabilities exist, which are associated with web services. Finally, the names of vendors and products extracted from the CPE are useful for classifying network services uniquely associated with the product. Apache Tomcat, for instance, is associated with web vulnerabilities, whereas Google Chrome is associated with client initiated vulnerabilities.

We selected a decision-tree classifier using the features from the CVSS, CPE, and CWE described above and a Gini-index for branching. We labeled each vulnerability from the NVD with a network service. The samples were shuffled and partitioned into 80% for training and 20% for testing. Classification results are shown in Table 1. Machine learning has a very high performance of 99% precision. Notably, 95% of vulnerabilities are either client initiated or related to web services. In some cases, the model has a high precision for other network services, but the number of support samples is too low for reliable use.

Table 1. Decision Tree Classification Report for Network Services

Network Service	Classification Summary			
	Precision	*Recall*	*F1-score*	*Support*
client	0.99	0.99	0.99	3044
web	0.99	0.99	0.99	650
mysql	1	1	1	53
smb	0.73	0.73	0.73	30
virtual	0.79	1	0.88	30
unknown[a]	0.32	0.32	0.32	19
dns	0.6	0.33	0.43	9
rdp	0.5	0.38	0.43	8

[a]NVD does not provide enough information to classify.

While this seems hopeful for the sole use of the decision-tree classifier, the remaining approximately 5% of the vulnerabilities include mostly remote attack vectors on other commonly used network ports. Missed classification of the remaining vulnerabilities undermines the overall use of the model. Therefore, to ensure ultra-high precision accuracy, the rejected samples feed into the NLP model for more accurate classification.

5.2 Natural Language Processing

Vulnerability descriptions are used to automate the identification of rejected machine learning samples. The vulnerability descriptions do not directly and consistently refer to a standard port number of network service. Instead, we have to develop more context from which to derive these features by training new categories through named entity recognition (NER). These models use an open-source NLP development tool spaCy and the associated Prodigy annotation tool [13]. In our approach to NLP, we begin with the identification of generic words or parts of speech categories from which humans can infer information about the affected network service. Then we use the new entities to build sentence recognition patterns used for deterministic identification of the network service. In effect, we model the way a security analyst identifies the affected network services from vulnerability summary statements.

First of all, it may be possible for the vulnerability description to name the network service. Several descriptions refer to the network service or port number

when discussing how to exploit the vulnerability (i.e., via HTTP) or the applicability of the vulnerability (i.e., a server running terminal services). Here, we construct a NER category of NETWORK SERVICE, which may not identify the vulnerable service but could provide additional context to the vulnerability description in the form of a new category of nouns.

Next, we observe the type of attack associated with the vulnerability can often implicate the network service directly or provide additional context. For example, one of the most common categories of vulnerabilities includes cross- site scripting, and terms such as XSS, Cross-Site Request Forgery (CSRF) and Server-Side Request Forgery (SSRF) provide enough context to determine the vulnerability is associated with a web service. We refer to this NER category as NETWORK ATTACK.

The given software may also provide enough context to determine the server port. A vulnerability in Microsoft Terminal Services is enough to identify the associated listening server over TCP port 3389. Likewise, a description of HTTP associated with a Chrome browser provides enough context to know the vulnerability is not associated with a listening service. We label this new category as NETWORK SOFTWARE.

Finally, operating system vulnerability descriptions typically refer to the vulnerable software component, NETWORK COMPONENT. Several variations of Linux, for example, refer to Firefox or Thunderbird as the noun in a prepositional phrase describing the vulnerable component. In large software packages, the vulnerable component could assist in developing rules for identifying the affected network service.

To test this approach, we annotated 2,363 vulnerability descriptions from the years 2017-2019 NVD data set. Extending the neural-network training model provided in the spacy application[1], we performed 10 training passes with a 20% test split. Table 2 shows the prediction accuracy results. These results show significant variance in the accuracy of the NER category, which we expect because the category of network attack and network service is more specific than the broad category of software.

Table 2. NLP NER Recognition Scores over the 2017-2019 NVD Dataset

NLP	NER Results Summary		
NER Category	*Precision*	*Recall*	*F1-score*
NETWORK SERVICE	0.72	0.62	0.66
NETWORK ATTACK	0.85	0.80	0.82
NETWORK SOFTWARE	0.60	0.61	0.61
NETWORK COMPONENT	1.00	0.47	0.64

The NER models for NETWORK SERVICE, NETWORK ATTACK, NET*WORK SOFTWARE*, and *NETWORK COMPONENT* further define speech categories in the vulnerability descriptions, but this is not enough to associateto a network service. To go from the general vulnerability-related speech categories to specific network services, we develop a set of patterns to match againstthe categorized vulnerability description. Each pattern match maps to a singlenetwork service. As a simple example, if the NER text for a network service is *mysql*, then we assign the network service MySQL-service to the vulnerability. Algorithm 3 in appendix C shows this approach in which the vulnerability description passes as input. We sort the patterns by accuracy, and greedily return the highest pattern matching the description.

The accuracy of the result is assessed back in the main algorithm and rejectedif the accuracy score falls below μ. A manual categorization of vulnerabilitydescriptions to network services assesses the accuracy of a given set of NLP patterns, as shown in the main algorithm 2.

Through this process, identified network services can then map to numeric network ports using a list of commonly associated port numbers from IETF standards, IANA port registrations, and vendor descriptions. This port mapping is not necessarily mutually exclusive and can be applied broadly to minimize the risk of missing an open port to a vulnerability.

6. Implementation

To test the proposed approach and preserve the confidentiality requirements for organizations participating in our research, we use a sample system developedto study the organizational management of vulnerabilities. The sample systemis described further in appendix B. The 2019 NVD dataset is extracted and mapped to our sample data set, and then we run our model with the combined data. In this experiment, the data allows us to assess approaches for including network topology and the types of adversaries to incorporate.

6.1 Summary of Results

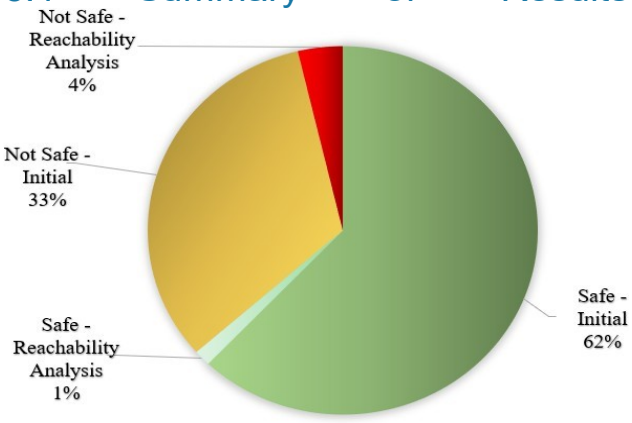

Fig 3. Vulnerability Assessment on Sample Dataset for 2019

Figure 4 shows the annual results for our sample dataset. In the experiment, we performed the initial

safety assessment using vulnerability dominance before assessing network reachability. In this, we found 5,467, or 62% of the vulnerabilities were considered safe due to the attack vector networking label.

$$v^{AV} > \tau^{AV} \implies (v)dom(\epsilon, \tau) \quad (6)$$

An additional 24 vulnerabilities were considered safe based on the user interaction label.

$$v^{UI} > \tau_{UI} \implies (v)dom(\epsilon, \tau) \quad (7)$$

The remaining 38% of vulnerabilities primarily represent server device vulnerabilities requiring further assessment. For these vulnerabilities, we extracted network service features using the AI pipeline and identified 33% as vulnerabilities associated with *client* (i.e., user-interactive) network services. All of these were on operator workstations or jump hosts, which we assume use automatic patching. Therefore, these are of less interest because they do not fall in the *server* category of assets.

The remaining 5% of vulnerabilities account for *server* vulnerabilities witha network attack-vector. For these, we determined 1% would be considered safebased on the types of network services impacted. For example, we assume a Windows domain controller would not run a web server. The remaining 4% wouldbe considered potentially unsafe. These could reduce further through reachability assessment by analyzing the network topology and firewall policy, which we leaveas future work to define a realistic firewall policy for our sample data set.

In total, for a year of applicable vulnerabilities, we consider 63% safe, 33% involving *client* network service vulnerabilities, and a maximum of 4% provided back to the security analyst for further evaluation

7. Conclusion

We have demonstrated a new approach by which organizations may assess their risk of applicable vulnerabilities, thereby freeing them to apply a more efficient strategy to vulnerability mitigation. The approach involves (i) extracting net- work service information using artificial intelligence, (ii) assessing the safety of applicable vulnerabilities using a cumulative set hierarchy and dominating set function, and (iii) tying firewall topology data to the applicable vulnerability. Using this model, we have shown the potential to focus vulnerability mitigation on at most 4% of all vulnerabilities in a representative system.

Furthermore, we have provided algorithms to perform the network topology and safety assessment in linear time to support the possibility of real-time vulnerability safety assessment, alerting, and firewall configuration recommendations. One other application might be to alert firewall administrators of potentially unsafe configuration before applying a firewall policy.

References

[1] Common product enumeration standard. https://nvd.nist.gov/products/cpe, accessed: 2020-01-28

[2] Common vulnerability scoring system specification. https://www.first.org/cvss/ v3.1/specification-document, accessed: 2020-01-28

[3] Common vulnerability scoring system v3.1: Specification document. https://www.first.org/cvss/v3.1/specification-document, accessed: 2020-02-01

[4] Common weakness enumeration. https://cwe.mitre.org/, accessed: 2020-01-28

[5] Exploitdb feed. https://www.exploit-db.com/, accessed: 2020-02-01

[6] Kitploit exploit collector. https://exploit.kitploit.com/, accessed: 2020-02-01

[7] National vulnerability database data feed. https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time, accessed: 2020-02-01

[8] National vulnerability database data feed. https://nvd.nist.gov/vuln/data-feeds, accessed: 2020-01-28

[9] North american electric reliability corporation (nerc) critical infrastructure protection (cip) standards. https://www.nerc.com/pa/Stand/Pages/CIPStandards.aspx, accessed: 2020-02-01

[10] Np live by network perception. https://www.network-perception.com/np-live/, accessed: 2020-02-02

[11] Packetstorm security database. https://packetstormsecurity.com/files/tags/ exploit/, accessed: 2020-02-01

[12] Rapid7 metasploit database. https://www.rapid7.com/db/?type=metasploit, accessed: 2020-02-01

[13] spacy and prodigy network language processing tools. https://explosion.ai/, accessed: 2020-02-02

[14] Vulners database. https://vulners.com/search?query=order:published, accessed: 2020-02-01

[15] Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. p. 217–224. New York, NY, USA (2002)

[16] 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). https://doi.org/10.1109/CSF.2018.00012

[17] Collins, K.: The hackers who broke into equifax exploited a flawin open-source server software. Quartz https://qz.com/1073221/ the-hackers-who-broke-into-equifax-exploited-a-nine-year-old-security-flaw/

[18] Fila, B., Wide-l, W.: Efficient attack-defense tree analysis using pareto attribute domains. In: 2019 IEEE 32nd Computer Security Foundations Symposium (CSF) (June 2019)

[19] Gamarra, M., Shetty, S., Nicol, D.M., Gonzalez, O., Kamhoua, C.A., Njilla, L.: Analysis of stepping stone attacks in dynamic vulnerability graphs. In:

2018 IEEE International Conference on Communications (ICC). pp. 1–7 (May 2018)
[20] Ghosh, N., Ghosh, S.K.: An approach for security assessment of network con- figurations using attack graph. In: Proceedings of the 2009 First International Conference on Networks Communications. p. 283–288. NETCOM '09, IEEE Computer Society, USA (2010). https://doi.org/10.1109/NetCoM.2009.83, https://doi.org/10.1109/NetCoM.2009.83
[21] Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack– defense trees. In: International Workshop on Formal Aspects in Security and Trust.pp. 80–95. Springer (2010)
[22] Landwehr, C.E.: Formal models for computer security. ACM Computing Surveys (CSUR) **13**(3), 247–278 (1981)
[23] 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)
[24] 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. IEEE (2019)
[25] 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)
[26] Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Information Security and Cryptology - ICISC 2005. pp. 186–198. Springer Berlin Heidelberg (2006)
[27] Noel, S., Jajodia, S.: Metrics suite for network attack graph analytics. In: Proceedings of the 9th Annual Cyber and Information Security Research Conference. p. 5–8. CISR '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2602087.2602117, https://doi.org/10.1145/ 2602087.2602117
[28] 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)
[29] Schneier, B.: Attack trees. Dr. Dobb's journal **24**(12), 21–29 (1999)
[30] Wang, L., Jajodia, S., Singhal, A., Noel, S.: k-zero day safety: Measuring the security risk of networks against unknown attacks. In: European Symposium on Research in Computer Security. pp. 573–587. Springer (2010)
[31] 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. IEEE (2019)
[32] Wing, J.M., et al.: Scenario graphs applied to network security. Information assurance: survivability and security in networked systems pp. 247–277 (2008)
[33] 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 - Volume 01. p. 427–431. MINES '09, IEEE Computer Society, USA (2009). https://doi.org/10.1109/MINES.2009.136, https://doi.org/10.1109/MINES.2009.136
[34] Xu, M., Huber, M., Sun, Z., England, P., Peinado, M., Lee, S., Marochko, A., Mat-toon, D., Spiger, R., Thom, S.: Dominance as a new trusted computing primitive for the internet of things. In: 2019 IEEE Symposium on Security and Privacy (SP)(2019)
[35] Yamamoto, Y., Miyamoto, D., Nakayama, M.: Text-mining approach for estimating vulnerability score. In: 2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS). pp. 67–73 (Nov 2015)

Appendix A Proof of Vulnerability Safety

Proof. We begin by assuming the dominance relation. Each capability-based category forms an ordered set which is also a cumulative set hierarchy with respect to v, ϵ and τ .
$\forall d \in D \mid 0 \leq i < |d|$,
$v^{d}_{i} \subseteq v^{di+1}$, $\epsilon^{di+1} \subseteq \epsilon^{di}$, $\tau^{di+1} \subseteq \tau^{di}$
Recall the set is ordered in terms of increasing difficulty or decreasing opportunity of exploit. For v, lower ordered categories are a subset of those higher ordered meaning that in relation to ϵ and τ , capability to exploit a higher ordered labels would also apply to any of those lower ordered in the category. 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 $d \in D$ in which v is greater than either ϵ or τ . Through the cumulative set hierarchy, it follow then that: . Through the cumulative set hierarchy, it follow then that:
$\exists d \in D \mid v^{d} \cap \epsilon^{d} \in \emptyset \vee v^{d} \cap \tau^{d} \in \emptyset$ (8)
Therefore, the capability does not exist for the adversary to exploit the vulnerability on the target in at least one category, and by definition 2, v is safe.
The converse is not necessarily true because there may be some other capability category not accounted for in the CVSS.

Appendix B Sample Data Set

The sample data set is derived from a mid-sized control center for the North American power grid. We derive the data using the computing environment required by the North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) regulatory standards [9]. The standard sufficiently references specific types of technology related to security for creating a representative sample. We supplement this with additional electric power control system components experientially inferred. As a result, we have a data set of 124 computing assets organized into 25 asset groups as shown in Figure 5. In each of the asset groups, we categorize the features used for comparison with the vulnerability CVSS features. This

includes User-Interaction, Confidentiality Impact, Integrity Impact and Availability Impact.

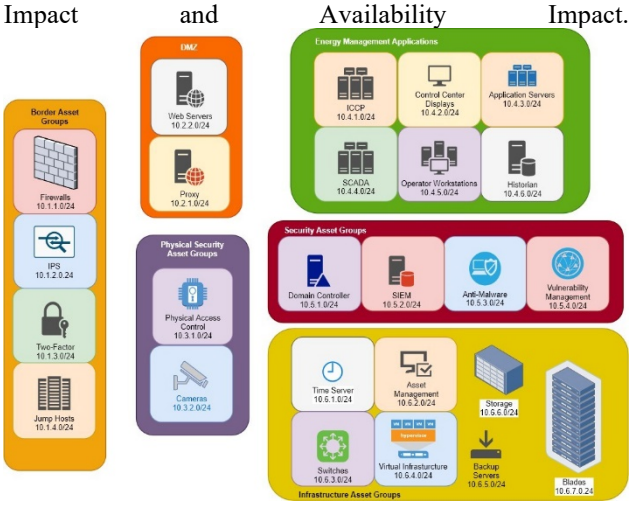

Fig. 4. Sample Data Set Asset Groups

The data set comprises of 4,894 combined operating system and software assets each mapping to a Common Product Enumerator (CPE) in the NVD. For example, the web server asset Primary-Proxy contains a CPE mapping cpe:2.3:a:apache:http server to represent the installed Apache Webserver. Most of the software mappings come from servers having a base install of Windows, Red Hat, or CentOS, but we also include device mappings for assets such as RTUs, network switches, firewalls and cameras, to name a few.

Through the CPE mappings, we obtain the vulnerabilities applicable to the sample environment from NVD for the year 2019. Overall, this provides 8,839 applicable vulnerabilities. The NVD does not provide the temporal exploitability metrics, which we need for assessing dominance with respect to a given adversary. This is provided through a licensed Vulners feed, which in turn aggregates exploitability metrics from Exploit DB, Metasploit, Packetstorm and Kitploit [14,5,12,6,11].

The data features used in the safety checking and machine learning include the NVD provided CVSS string and our sample set asset group features (i.e. the CVSS related-features). The network topology generation from firewall configuration files is provided by NP Live. However, we do not have firewall configuration data for the sample data set. Instead, we approximate the reachability through manual inspection.

Appendix C NLP Accuracy

The accuracy of the result is assessed back in the main algorithm and rejected if the accuracy score falls below μ. A manual categorization of vulnerability descriptions to network services is used to assess the accuracy of a given set of NLP patterns as shown in the main algorithm 2.

Algorithm 3 NLP Pattern Matching Algorithm

1: Use the trained model NLP with the new NER categories
2:The Patterns object contains both an NLP pattern and accuracy score
3: **function** NET NLP(description)
4:Use the trained modelNLP with the new NER categories
5:*Ents* ← *NLP* (*description*)
6: **for all** $p \in Patterns$ **do**
7: **if** *p.match*(*Ents*) **then**
8: **return** *p*
9: **end if**
10: **end for**
11: **return** Uncategorized
12: **end function**

To build and maintain the accuracy score for the pattern matching, a modestlevel of human work is required. We describe this process in algorithm 4. As patterns are matched, human inspection builds the accuracy score necessary for subsequent automated processing of the NLP result. Humans can perform this process manually and centralized as a service. We can also maintain the performance of rules. As shown on line 12, as the accuracy score drops below a threshold, the pattern is removed from the model. Likewise, if no pattern is found, then the description is flagged for the development of a new pattern.

Algorithm 4 Maintain NLP

1: **function** NLP-Maintain(description)
2: Use the trained model NLP
3: *Ents* ← *NLP(description)*
4: **for all** $p \in Patterns$ **do**
5: **if** *p.match(Ents)* **then**
6: *Match = T rue*
7: $p.samples \leftarrow p.samples + 1$
8: **if** Correct match **then**
9: $p.correct \leftarrow p.correct + 1$
10: **else if** Incorrect match **then**
11: Analyze and modify pattern
12: **end if**
13: $p.score = \frac{p.correct}{p.samples}$
14: **if** p.score < score min **then**
15: *p.delete()*
16: *Match = False*
17: **end if**
18: **end if**
19: **end for**
20: **if** ¬*Match* **then**
21: Flag description for new pattern, new
22: Patterns.add*(new)*
23: **end if**
24: Sort patterns by descending accuracy
25: **return** Sort(*Patterns*, desc)
26: **end function**