

# Vulnerability Metrics for Graph-based Configuration Security

Ibifubara Iganibo<sup>1</sup><sup>a</sup>, Massimiliano Albanese<sup>1</sup><sup>b</sup>, Marc Mosko<sup>2</sup><sup>c</sup>, Eric Bier<sup>2</sup>  
and Alejandro E. Brito<sup>2</sup>

<sup>1</sup>Center for Secure Information Systems, George Mason University, Fairfax, U.S.A.

<sup>2</sup>Palo Alto Research Center, Palo Alto, U.S.A.

**Keywords:** Configuration Security, Vulnerability Analysis, Vulnerability Graphs, Metrics.

**Abstract:** Vulnerability analysis has long been used to evaluate the security posture of a system, and vulnerability graphs have become an essential tool for modeling potential multi-step attacks and assessing a system's attack surface. More recently, vulnerability graphs have been adopted as part of a multi-faceted approach to configuration analysis and optimization that aims at leveraging relationships between the components, configuration parameters, and vulnerabilities of a complex system to improve its security while preserving functionality. However, this approach still lacks robust metrics to quantify several important aspects of the system being modeled. To address this limitation, we introduce metrics to enable practical and effective application of graph-based configuration analysis and optimization. Specifically, we define metrics to evaluate (i) the exploitation likelihood of a vulnerability, (ii) probability distributions over the edges of a vulnerability graph, and (iii) exposure factors of system components to vulnerabilities. Our approach builds upon standard vulnerability scoring systems, and we show that the proposed metrics can be easily extended. We evaluate our approach against the Common Weakness Scoring System (CWSS), showing a high degree of correlation between CWE scores and our metrics.

## 1 INTRODUCTION


For almost two decades, graph-based vulnerability analysis has been used to evaluate the security posture of a system, and vulnerability graphs have become an essential tool for modeling potential multi-step attacks and assessing a system's attack surface (Ammann et al., 2002; Jajodia et al., 2005).


More recently, vulnerability graphs have been adopted as part of a multi-layer graph approach to configuration analysis and optimization, referred to as SCIBORG (Soroush et al., 2020). This multi-faceted approach aims at modeling relationships between the components, configuration parameters, and vulnerabilities of a complex system by ingesting data from a number of different data sources, including but not limited to system documentation, operating procedures, and reports from vulnerability scanners. The resulting graph model is then analyzed with the goal of improving the security of the system while preserving its functionality. Ultimately, SCIBORG


generates a detailed report containing, among other valuable pieces of information, a list of recommended configuration changes.

As cyber systems are becoming more complex and connected, configuration analytics and optimization are becoming increasingly critical for their correct and secure operation. Attackers usually rely on unpatched vulnerabilities and configuration errors to gain unauthorized access to system resources. Misconfiguration can occur at any level of a system's software architecture, and correctly configuring systems becomes more complex when many interconnected systems are involved. In 2017, Security Misconfiguration was listed by OWASP amongst the ten most critical web application security risks (owa, 2017). Most current configuration security approaches focus on tuning the configuration of individual components, but lack a principled approach to managing the complex relationships between the configuration parameters of the components of a composed system.

SCIBORG's approach tackles this problem by building upon previous work on impact analysis of multi-step attacks (Albanese and Jajodia, 2018) to determine attack paths enabled under a given system

<sup>a</sup> <https://orcid.org/0000-0003-1321-8554>

<sup>b</sup> <https://orcid.org/0000-0002-2675-5810>

<sup>c</sup> <https://orcid.org/0000-0002-3270-8738>

configuration. However, it still lacks robust metrics to quantify several important aspects of the system being modeled and effectively assess the security implications of configuration choices.

To address this limitation and enable practical and effective application of configuration analytics and optimization, we introduce metrics to augment vulnerability graphs, which are a critical component of the graph models presented in (Albanese and Jajodia, 2018) and (Soroush et al., 2020). Specifically, we define the *exploitation likelihood* (or simply *likelihood*) of a vulnerability as the probability that an attacker will attempt to exploit that vulnerability, if certain preconditions are met. Specific preconditions may vary depending on the characteristics of each vulnerability, as certain configuration settings may prevent access to vulnerable portions of the target software.

We identify several variables that influence an attacker’s decision to exploit a given vulnerability, but our model is general enough to account for additional variables that one may deem relevant. We then define probability distributions over the edges of a vulnerability graph – to model how an attacker may select the next target exploit in a multi-step attack – and exposure factors of system components to vulnerabilities. We propose metrics to quantify these concepts, building upon standard vulnerability scoring systems. The proposed metrics have been evaluated against the Common Weakness Scoring System, showing a high degree of correlation.

In summary, the key contributions of this paper are: (i) a general and extensible formal approach to assess the likelihood that an attacker will attempt to exploit a vulnerability as well as the impact that a successful exploitation would entail; (ii) the use of Intrusion Detection System (IDS) rules in the computation of both likelihood and impact; and (iii) a set of metrics to complement graph models built around vulnerability graphs, including but not limited to the multi-layer graphs generated by SCIBORG.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 summarizes current standards in vulnerability scoring and provides a brief overview of IDS rules. Next, Section 4 gives an overview of the SCIBORG model introduced in (Soroush et al., 2020). Then, Section 5 introduces the proposed vulnerability metrics, and Section 6 reports on the results of our evaluation. Finally, Section 7 gives some concluding remarks and outlines possible future research directions.

## 2 RELATED WORK

The ultimate goal of configuration analytics and optimization is to reduce a system’s attack surface. Thus, the ability to measure a system’s susceptibility to attacks is critical to achieve this goal. Over the years, researchers and practitioners have proposed various attack surface measurement techniques and metrics (Bopche et al., 2019; Manadhata and Wing, 2011; Stuckman and Purtilo, 2012; Yoon et al., 2020). However, these techniques do not typically consider the relationships between system components, vulnerabilities, and configurations. This limitation prevents existing metrics from accurately measuring a system’s attack surface. An inaccurate assessment of a system’s susceptibility to attacks may then result in adopting inadequate countermeasures, which can have devastating effects on the overall security posture of the system.

Furthermore, to develop comprehensive cyber situational awareness (Jajodia and Albanese, 2017), and in line with more traditional risk analysis approaches, one has to distinguish between the likelihood that a vulnerability might be exploited and the impact an actual exploitation would cause. Most current approaches provide overall scores that, while possibly considering multiple variables, do not involve probabilistic considerations. Furthermore, even the approaches that consider the effect of multiple variables on the overall scores assigned to vulnerabilities cannot be easily extended if one wants to consider additional variables that were not originally taken into account – such as the *age* of a vulnerability and the set of IDS rules associated with it – and do not allow one to change the relative weights of these variables. Other recent metrics (Mukherjee and Mazumdar, 2018; Wang et al., 2019) use scores from the Common Vulnerability Scoring Systems (CVSS) or the Common Weakness Scoring Systems (CWSS) in isolation or as the dominant factor in determining the severity of a vulnerability.

The approach we propose in this paper addresses these limitations by creating a general and extensible system of metrics that builds upon existing literature on vulnerability graphs and vulnerability scoring.

## 3 TECHNICAL BACKGROUND

The vulnerability metrics presented in this paper rely on information from the National Vulnerability Database (NVD) and on scores computed through the Common Vulnerability Scoring System (CVSS). We also rely on repositories of Intrusion Detection Sys-

tem (IDS) rules in the computation of our likelihood and impact metrics. While our approach is general and does not require the adoption of a specific intrusion detection system, we have used Snort<sup>1</sup> and Suricata<sup>2</sup> in our implementation and experimental evaluation. The proposed metrics have then been validated against the Common Weakness Scoring System (CWSS). This section provides a brief overview of NVD, CVSS, CWSS, and IDS rules.

### 3.1 NVD

The National Vulnerability Database (NVD) is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol (SCAP), and is maintained by the National Institute of Standards and Technology (NIST). This data enables automation of vulnerability management, security measurement, and compliance.

NVD is built upon and fully synchronized with the Common Vulnerabilities and Exposures (CVE) List, that is a list of records including an identification number, a description, and public references for publicly known cybersecurity vulnerabilities. The CVE repository is maintained by MITRE, and NVD augments it with severity scores, and impact ratings based on the Common Vulnerability Scoring System.

### 3.2 CVSS

As stated by the Forum of Incident Response and Security Teams (FIRST) – which currently maintains it – the Common Vulnerability Scoring System (CVSS) provides a means to “capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity”. This score is calculated based on three different metrics:

- Base Score Metrics (required)
- Temporal Score Metrics (optional)
- Environmental Score Metrics (optional)

CVSS is currently at version 3.1, but, for our analysis, we use CVSS version 2.10, as scores are available for a larger number of vulnerabilities. Additionally, we only consider Base Score Metrics in our analysis, as the other metrics are not yet widely used. The Base Score is computed via Equation 1 below.

$$BaseScore = (0.6 \cdot I + 0.4 \cdot E - 1.5) \cdot f(I) \quad (1)$$

where  $I$  and  $E$  are the Impact and Exploitability scores defined by Equations 2 and 3 respectively, and  $f(I)$

<sup>1</sup><https://www.snort.org/>

<sup>2</sup><https://suricata-ids.org/>

is defined by Equation 4. The Impact score quantifies the consequences of an exploit, whereas the Exploitability score captures how easy to exploit a vulnerability is.

$$I = 10.41 \cdot (1 - (1 - I_C) \cdot (1 - I_I) \cdot (1 - I_A)) \quad (2)$$

$$E = 20 \cdot AC \cdot A \cdot AV \quad (3)$$

$$f(I) = \begin{cases} 0, & \text{if } I = 0 \\ 1.176, & \text{otherwise} \end{cases} \quad (4)$$

The  $I_C$ ,  $I_I$ , and  $I_A$  scores in Equation 2 are the confidentiality, integrity, and availability impact respectively, as defined in Table 1.

Table 1: Impact Metrics.

	Confidentiality Impact ( $I_C$ )	Integrity Impact ( $I_I$ )	Availability Impact ( $I_A$ )
None	0.000	0.000	0.000
Partial	0.275	0.275	0.275
Complete	0.660	0.660	0.660

The  $AC$ ,  $A$ , and  $AV$  scores in Equation 3 are the exploitability metrics Access Complexity, Authentication, and Access Vector, as defined in Table 2.

Table 2: Exploitability Metrics.

Access Compl. ( $AC$ )	Authentication ( $A$ )	Access Vector ( $AV$ )	
High	Multiple	Local	0.395
Medium	Single	Adjacent	0.646
Low	None	Network	1.000

### 3.3 CWE and CWSS

Common Weakness Enumeration (CWE) is a system that provides a structured list of clearly defined software and hardware weaknesses<sup>3</sup>. A software weakness is not necessarily a vulnerability, but weaknesses may become vulnerabilities. MITRE’s Common Weakness Scoring System (CWSS) provides a mechanism for prioritizing software weaknesses that are present within software applications in a consistent and flexible manner<sup>4</sup>. It is a collaborative, community-based effort that is addressing the needs of its stakeholders across government, academia, and industry.

CWSS is organized into three metric groups: Base Finding, Attack Surface, and Environmental. Each group includes multiple metrics – also known as factors – that are used to compute a CWSS score for a weakness. While discussing the formulation of these

<sup>3</sup><https://cwe.mitre.org/>

<sup>4</sup><https://cwe.mitre.org/cwss/>

metrics goes beyond the scope of this paper and we refer the reader to the documentation for further details, in the following we focus our attention on the method MITRE used to rank the most dangerous weaknesses. In Section 6, we rely on this approach to validate our metrics. Equation 5 defines the set of CVEs mapped to a given CWE, and Equation 6 defines the number of times each CWE is mapped to CVE entries<sup>5</sup>.

$$C(CWE_i) = \{CVE_j \in NVD, CVE_j \rightarrow CWE_j\} \quad (5)$$

$$Freqs = \{|C(CWE_i)|, CWE_i \in NVD\} \quad (6)$$

Then Equations 7 and 8 respectively compute the frequency and severity of a CWE, where the severity is based on the average CVSS score. Both the frequency and severity are normalized between 0 and 1.

$$Fr(CWE_i) = \frac{|C(CWE_i)| - \min(Freq)}{\max(Freq) - \min(Freq)} \quad (7)$$

$$Sv(CWE_i) = \frac{avg_{CWE_i}(CVSS) - \min(CVSS)}{\max(CVSS) - \min(CVSS)} \quad (8)$$

Finally Equation 9 defines the overall score of a CWE as the product of its frequency and severity, normalized between 0 and 100.

$$Score(CWE_i) = Fr(CWE_i) \cdot Sv(CWE_i) \cdot 100 \quad (9)$$

### 3.4 IDS Rules

In our approach, we use Intrusion Detection System (IDS) rules as one of the factors influencing the computation of both the likelihood of a vulnerability exploit and the exposure factor of a system component to a vulnerability. However, we distinguish between *known* and *deployed* rules, and only consider rules that are explicitly mapped to CVE entries. We use the term *known IDS rule* to refer to any IDS rule that is available to the community through publicly accessible repositories. Our assumption is that the existence of known IDS rules associated with a given vulnerability may decrease the likelihood of exploiting that vulnerability, as an attacker may prefer to target vulnerabilities that can be exploited without triggering IDS alerts. As an example, Figure 1 shows the results of searching the Snort rule repository for two different CVEs, that is CVE-2018-12572 and CVE-2018-12572 respectively.

<sup>5</sup>We slightly abuse notation and use  $CWE_i \in NVD$  to denote a CWE that is mapped to at least one CVE entry in NVD.

Summary: 'CVE-2018-11776'	
Search returned 7 results	
1-29639	SERVER-APACHE Apache Struts wildcard matching OGNL remote code execution attempt
1-39190	SERVER-APACHE Apache Struts remote code execution attempt
1-39191	SERVER-APACHE Apache Struts remote code execution attempt
1-47634	SERVER-APACHE Apache Struts OGNL getRuntime.exec static method access attempt
1-47689	SERVER-APACHE Apache Struts javanet.Socket class access attempt
1-47690	SERVER-APACHE Apache Struts javalang.ProcessBuilder class access attempt
1-47691	SERVER-APACHE Apache Struts ognl remote code execution attempt

Summary: 'CVE-2018-12572'	
Search returned 1 results	
1-49325	FILE-OTHER Microsoft Windows Avast Anti-Virus local credentials disclosure attempt

Figure 1: Snort rules associate with different CVEs.

Instead, we use the term *deployed IDS rule* to refer to any IDS rule that is being actively used by a deployed IDS. Deployed rules may include a subset of known rules or ad hoc rules developed by the system’s administrators. An attacker may not be aware of what IDS rules are actually in use, so these rules do not affect the likelihood of exploiting a vulnerability. However, early detection of intrusions may help mitigate the consequences of an exploit, therefore we take deployed rules into account in the computation of exposure factors.

## 4 THE SCIBORG MODEL

This section describes the SCIBORG graph model that was presented in (Soroush et al., 2020). The metrics we introduce in this paper are designed to address previous limitations of this model by adding the capability of quantifying several important aspects of the systems being modeled. However, we remark that our approach is not limited to graphs generated by SCIBORG, and is applicable to any graph model that, like SCIBORG, relies on vulnerability graphs.

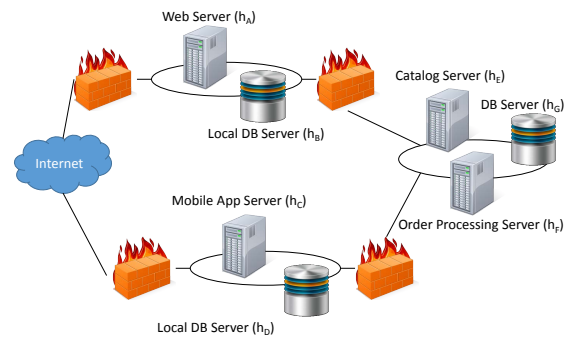


Figure 2: Network diagram of a notional distributed system.



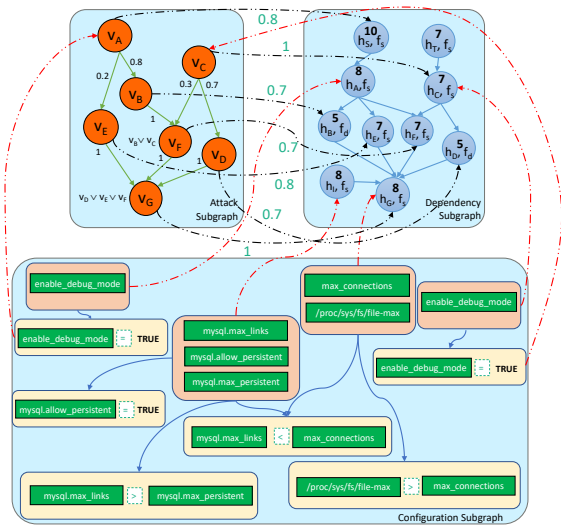


Figure 3: The SCIBORG graph for the system of Figure 2.

SCIBORG’s approach is based on modeling a distributed system as a three-layer directed graph encoding all the information needed to reason upon the optimality of system configurations. The three layers, described in more detail in the following subsections, are (i) a dependency subgraph; (ii) a vulnerability subgraph; and (iii) a configuration subgraph. For illustrative purposes, a three-layer graph corresponding to the notional distributed system of Figure 2 is depicted in Figure 3.

### 4.1 Dependency Subgraph

Configuration changes in one component can impact the security and functionality of other components. Therefore, globally optimal security decisions need to rely on dependency information. Several approaches have been proposed to discover implicit or undocumented dependencies (Bahl et al., 2006; Natrajan et al., 2012), and SCIBORG can ingest dependency information generated through multiple tools.

A node in the dependency subgraph represents a system component (host, service, etc.), and a directed edge represents a dependency between two components. When dependencies are accurately captured, dependency graphs are expected to be acyclic. To capture a wide range of relationships between components, SCIBORG models each dependency as a function of the form  $f : [0, 1]^n \rightarrow [0, 1]$ , with  $f(0, \dots, 0) = 0$  and  $f(1, \dots, 1) = 1$ .

Each component has an intrinsic utility for the owning organization and its *dependency function* defines its ability to provide its expected utility, based on the status of the components it depends on: the arguments of this function are the percentage resid-

ual utilities of those components and are in turn computed through each component’s respective dependency function. A dependency function returns 1 when the component can provide 100% of its utility, and 0 when it has been completely compromised.

Three types of dependency relationships are identified in (Soroush et al., 2020), namely *redundancy* ( $f_r$ ), *strict dependence* ( $f_s$ ), and *graceful degradation* ( $f_d$ ), but such classification is not intended to be exhaustive, and other dependency relationships can be defined.

Figure 3 includes the dependency subgraph for our notional system. An edge from  $h_A$  to  $h_B$  denotes that  $h_A$  depends on  $h_B$ . Each component node is labeled with the type of dependency and its utility. Utility values can be assigned by a domain expert or automatically derived by computing graph-theoretic centrality metrics (Kourtellis et al., 2015). In the security field, ad-hoc centrality measures have been used for botnet detection and mitigation (Venkatesan et al., 2015).

### 4.2 Vulnerability Subgraph

Vulnerability subgraphs, also known as attack graphs, are powerful conceptual tools to represent knowledge about vulnerabilities and their dependencies. SCIBORG adopts a broader definition of *vulnerability*, but for the purpose of the analysis presented in this paper we refer to vulnerability graphs as formalized in (Albanese and Jajodia, 2018). A node in the vulnerability subgraph represents a known vulnerability, and an edge between two vulnerabilities, referred to as an *ENABLES* edge, indicates that exploiting the first vulnerability creates the preconditions to exploit the second one. An edge from a node in the vulnerability subgraph to a node in the dependency subgraph, referred to as a *DEGRADES* edge, indicates that the exploitation of a given vulnerability can impact a component to an extent quantified by the exposure factor labeling that edge.

The vulnerability subgraph for our notional system is also depicted in Figure 3. This graph can be generated by combining information from network scanners (e.g., Nessus) and vulnerability databases (e.g., CVE, NVD), as shown in (Ammann et al., 2002; Jajodia et al., 2005). The edges in the vulnerability subgraph of Figure 3 are labeled with probabilities, which can be used to infer the most likely paths that an attacker might take in a multi-step attack. Determining these probabilities is an open research problem that we address in this paper, though useful heuristics exist (Albanese et al., 2013; Albanese and Jajodia, 2018), based on the assumption that vulnera-

bilities that require more resources, time, and skill are less likely to be exploited.

### 4.3 Configuration Subgraph

The configuration subgraph models relationships between configuration parameters, both within and across components of the composed system. There are two classes of nodes in this subgraph: *Class 1* nodes represent per-component configuration parameters, whereas *Class 2* nodes capture constraints on one or more configuration parameters. Edges from Class 1 nodes to a Class 2 node identify the parameters involved in a constraint. Directed edges from a component in the dependency subgraph to Class 1 nodes in the configuration subgraph identify the configuration parameters associated with that component.

Some of the constraints in the configuration subgraph may be specified in the system’s documentation. More importantly, some of the relationships between configuration parameters might enable or disable preconditions for vulnerabilities in one or more components. SCIBORG captures this information with directed edges from Class 2 nodes in the configuration subgraph to relevant nodes in the vulnerability subgraph.

### 4.4 Advantages and Limitations

SCIBORG’s approach differs from the traditional idea of minimizing the attack surface of a system by minimizing, for instance, the *number* of exploitable resources available to the adversary. Instead, SCIBORG focuses on minimizing the potential impact of possible attacks by analyzing the *paths* that an adversary can traverse in a multi-step attack that seeks to achieve a well-defined goal (e.g., exfiltrating sensitive information from a database), and evaluating the impact resulting from such attacks. Therefore, the focus is shifted from minimizing the number of vulnerable entry points to preventing or mitigating the attacks with the highest potential security impact.

However, SCIBORG still lacks a systematic approach to estimate the probabilities labeling edges in the vulnerability subgraph as well as the exposure factors labeling edges from vulnerability nodes to nodes in the dependency subgraph. In this paper, we address this limitation by proposing robust metrics to augment any model based on vulnerability graphs – including but not limited to SCIBORG – with the capability of quantifying different aspects of the systems being modeled.

## 5 VULNERABILITY METRICS

This section introduces the metrics we have developed to address current limitations of vulnerability graphs. For the purpose of this analysis, we assume that (i) the set of vulnerabilities in a system does not change over time<sup>6</sup>, although configuration changes might render some vulnerabilities not exploitable, and (ii) each vulnerability  $v$  corresponds to a CVE entry. The following subsections discuss in detail the different metrics that we have introduced.

### 5.1 Exploit Likelihood

As incidentally mentioned earlier, a vulnerability’s susceptibility to becoming a target for exploitation by a malicious user may depend on a number of variables, including features of the vulnerability itself and characteristics of potential attackers. While approaches considering the skills and resources available to different types of attackers have been explored in the literature (Leversage and Byres, 2008), such approaches are not useful in practice, as defenders should always operate under worst-case assumptions, and assume they are facing skilled and well-equipped attackers. Therefore, we focus our attention on features of the vulnerabilities themselves and on any information that may be available to potential attackers, irrespective of their skills, and that could influence their selection of target exploits.

We define the *exploitation likelihood* (or simply *likelihood*) of a vulnerability as the probability that an attacker will attempt to exploit that vulnerability, if given the opportunity. An attacker has the *opportunity* to exploit a vulnerability if certain preconditions are met, most notably if they have access to the vulnerable host. Specific preconditions may vary depending on the specific characteristics of each vulnerability, as certain configuration settings may prevent access to vulnerable portions of the target software. We argue that the following variables are the main factors influencing an attacker’s decision to exploit a given vulnerability.

- The vulnerability’s exploitability score, as captured by CVSS.

<sup>6</sup>In real-world scenarios, the vulnerability landscape might change for a number of reasons: applying patches, adding or removing software, or simply discovering previously unknown vulnerabilities. Generalizing our analysis to such a dynamic scenario would be relatively straightforward, and would involve treating various metrics and sub-metrics, such as the number  $|IDS_k(v)|$  of known IDS rules associated with vulnerability  $v$ , as functions of time.

- The amount of time since information about the vulnerability became public.
- The number of known Intrusion Detection System (IDS) rules associated with the vulnerability.

As discussed in Section 3, the CVSS Exploitability score captures how easy it is to exploit a vulnerability, based on different features captured by various metrics<sup>7</sup>, most notably Access Vector (AV) and Access Complexity (AC). The Access Vector metric reflects the context in which a vulnerability can be exploited. Its value is higher for vulnerabilities that can be exploited remotely, and are therefore more likely to be exploited as the number of potential attackers is larger than the number of potential attackers that could exploit a vulnerability requiring physical access to the vulnerable host. The Attack Complexity metric reflects the amount of effort and resources required for a successful attack. Its value is higher for exploits that require little or no effort, and are therefore more likely to be exploited.

The time passed since details about the vulnerability were made public also plays a role in determining the likelihood of exploitation. In fact, the longer a vulnerability has been known, the more exploits may have been developed by the hacker community. While it is true that the likelihood that patches have been developed also increases with time, it is well-known that patches are not applied promptly and consistently across systems, thus giving attackers a window of opportunity to target known but unpatched vulnerabilities.

Finally, the number of known IDS rules may influence the attacker's choice of vulnerabilities to exploit. With systems typically exposing multiple vulnerabilities, attackers may choose to avoid exploits that are more easily detectable.

Let  $G_v = (V, E)$  denote a vulnerability graph (e.g., the vulnerability subgraph of the SCIBORG model). Based on the considerations above, we define the exploitation likelihood as a function  $\rho : V \rightarrow [0, 1]$  defined as

$$\rho(v) = \frac{(1 - e^{-\alpha \cdot \sqrt{t(v)}}) \cdot (1 - e^{-\beta \cdot \text{Exploitability}(v)})}{e^{\gamma \cdot |\text{IDS}_k(v)|}} \quad (10)$$

where  $t(v)$  is the time since vulnerability  $v$  was discovered,  $\text{Exploitability}(v)$  is the the CVSS *Exploitability* score of  $v$ , and  $\text{IDS}_k(v)$  is the set of known IDS rules associated with  $v$ .

Each variable contributes to the overall likelihood as a multiplicative factor between 0 and 1 that is formu-

<sup>7</sup>Access Vector (AV) and Access Complexity (AC) are common across CVSS 2 and CVSS 3, whereas other exploitability metrics are specific to either version.

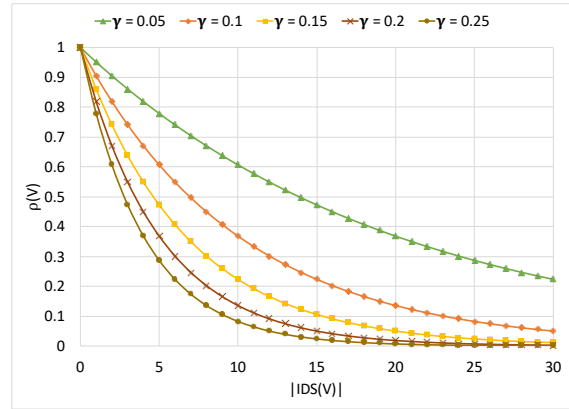


Figure 4: Effect of  $|\text{IDS}_k(v)|$  on the likelihood.

lated to account for *diminishing returns*. Factors corresponding to variables that contribute to increasing the likelihood are of the form  $1 - e^{-c \cdot f(x)}$ , where  $x$  is the variable,  $f(\cdot)$  is a function such that  $x_1 < x_2 \rightarrow f(x_1) < f(x_2)$ , and  $c$  is a constant. Similarly, factors corresponding to variables that contribute to decreasing the likelihood are of the form  $e^{-c \cdot f(x)} = \frac{1}{e^{c \cdot f(x)}}$ . This formulation provides several practical advantages: (i) the resulting likelihood is normalized between 0 and 1; (ii) accounting for the effect of additional independent variables would be straightforward; and (iii) ignoring the effect of a variable would simply entail setting the constant  $c$  such that the corresponding factor evaluates to 1 (i.e.,  $c = +\infty$  for factors increasing the likelihood and  $c = 0$  for factors decreasing the likelihood).

Figure 4 shows the effect of  $|\text{IDS}_k(v)|$  on the likelihood for various values of  $\gamma$ , assuming that all other factors evaluate to 1. As an example, for  $\gamma = 0.25$ , the existence of 3 known IDS rules associated with a vulnerability  $v$  cuts the likelihood of exploiting  $v$  approximately in half.

As for the function  $f(\cdot)$ , in most cases we define it as the linear function  $f(x) = x$ , but in the case of the time  $t$  since the vulnerability was disclosed, we use  $f(t) = \sqrt{t}$  to model a less-than-linear relationship, as suggested by Tripwire<sup>8</sup>.

## 5.2 Edge Probability

In the previous section, we defined the *exploitation likelihood* of a vulnerability as the probability that an attacker will attempt to exploit that vulnerability, *given the opportunity*. In other words, in computing the likelihood we do not take the context in which vulnerabilities are exploited into account. When at-

<sup>8</sup><https://www.tripwire.com/solutions/vulnerability-and-risk-management>

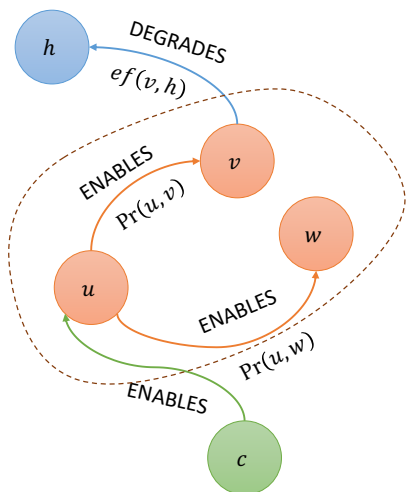


Figure 5: Example of edges in the SCIBORG graph model.

tempting to penetrate a complex networked system, attackers usually engage in multi-step attacks, which can be modeled through vulnerability graphs, as described in Section 4.2. At every step of the attack, adversaries will typically be able to choose among several vulnerabilities to exploit next in order to advance the attack. Therefore, for each node in the vulnerability subgraph, we need to compute a probability distribution over the outgoing ENABLES edges. As all the variables that can influence the attacker’s choice of vulnerabilities to exploit have been factored into each vulnerability’s likelihood, this probability distribution can be computed by normalizing the likelihood values of the *enabled* vulnerabilities and using the normalized values to label the corresponding ENABLES edges. Therefore, given an ENABLES edge  $e = (u, v)$  (shown in Figure 5), the probability of exploiting  $v$  after  $u$  is given by

$$\Pr(e) = \frac{\rho(v)}{\sum_{v' s.t. (u,v') \in E} \rho(v')} \quad (11)$$

The same reasoning can be applied to any ENABLES edge  $e = (c, u)$  between a constraint (Class 2 node) in SCIBORG’s configuration subgraph and a vulnerability.

### 5.3 Exposure Factor

Finally, the *exposure factor* (EF) – as typically defined in risk analysis terminology – represents the relative damage that an undesirable event – the exploitation of a vulnerability in our case – would cause to the affected asset. The single loss expectancy (SLE) of such an incident is then computed as the product between its exposure factor and the asset value (AV), that is  $SLE = EF \times AV$ .

Factors influencing the exposure factor include the CVSS impact score – which in turn considers the impact on confidentiality, integrity, and availability – and the number of deployed IDS rules that can potentially mitigate the consequences of an exploit. Formally, given a DEGRADES edge  $(v, h)$ , we define the exposure factor for this edge as

$$ef(v, h) = \frac{0.1 \cdot Impact(v)}{e^{\delta \cdot |IDS_d(v)|}} \quad (12)$$

Note that the notation  $ef(v, h)$  is redundant, as  $h$  is uniquely determined by  $v$ , which represents a specific instance of a known vulnerability that exists on a given component. If multiple components have the same vulnerability, this scenario is represented through multiple vulnerability nodes in the vulnerability subgraph, one per component.

## 6 EXPERIMENTAL EVALUATION

This section describes how we validated the proposed metrics against CWE. First, we ranked the 2020 CWE Top 25 Most Dangerous Software Weaknesses<sup>9</sup> using an ad hoc score based on our metrics but designed to be consistent with the logic behind the CWE score, therefore making the two rankings comparable. Second, we gathered and validated the data necessary to extend our analysis beyond the top 25 CWEs. Third, we compared our rankings of all the CWEs associated with 2018 and 2019 NVD data with rankings based on CWE scores, and showed a high degree of correlation. The experiments described in this section were run with  $\alpha = 0.75$ ,  $\beta = 0.25$ ,  $\gamma = 0$ , and  $\delta = 0$ , unless otherwise specified. We set  $\gamma = 0$  and  $\delta = 0$  to enable a meaningful comparison with CWE scores, which do not use information about IDS rules. However, we have already illustrated how IDS rules influence the likelihood (see Figure 4), and we further discuss the effect of considering IDS rules in Section 6.4.

### 6.1 Comparison Against Top 25 CWEs

MITRE published the 2020 CWE Top 25 list based on NVD data about vulnerabilities from years 2018 and 2019, which consists of approximately 27,000 CVEs that are associated with a weakness. Table 3 shows the 2020 CWE Top 25 with important scoring information. It includes the number of CVE entries mapped to a particular CWE and the average CVSS score for each weakness. Note that the CWE overall scores are normalized, as discussed in Section 3.3.

<sup>9</sup><https://cwe.mitre.org/top25/>



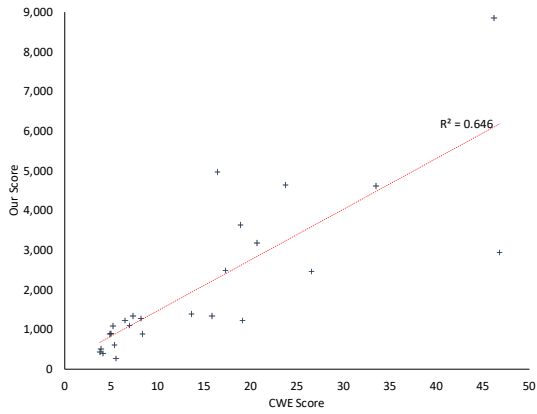


Figure 6: Correlation between MITRE’s overall scores and our scores for the 2020 CWE Top 25.

Table 4 shows again the 2020 CWE Top 25, but this time it includes the average likelihood and exposure factor for each CWE and the ad hoc score – hereinafter referred to as *our score* – we computed to compare our metrics against the CWE scores shown in Table 3. Our score of a CWE is defined as the product of the number of vulnerabilities mapped to that CWE, their average likelihood, and their average exposure factor. Differently from MITRE’s score, it is not normalized, but this is irrelevant for the purpose of analyzing the correlation. Note that the numbers of CVEs shown in Table 4 differ slightly from those in Table 3 for the reason explained in the following subsection. Our analysis shows a 80.3% correlation ( $R^2 = 0.646$ ) between the overall CWE scores in Table 3 and our scores in Table 4, as shown in Figure 6.

Table 3: Sample of 2020 CWE Top 25.

Rank	CWE ID	NVD Count	Avg CVSS	Score
1	CWE-79	3,788	5.80	46.82
2	CWE-787	2,225	8.31	46.17
3	CWE-20	1,910	7.35	33.47
4	CWE-125	1,578	7.13	26.50
5	CWE-119	1,189	8.08	23.73
...	...	...	...	...
25	CWE-862	236	6.90	3.77

Table 4: Sample of 2020 CWE Top 25 with our score.

Rank	CWE ID	No. CVEs	Average Likelihood	Average Exp. Factor	Our Score
1	CWE-787	2,012	6.02	0.73	8,850.36
2	CWE-78	777	7.29	0.88	4,974.04
3	CWE-119	1231	5.59	0.67	4,642.55
4	CWE-20	1,887	4.45	0.55	4,619.00
5	CWE-416	934	5.69	0.68	3,622.81
...	...	...	...	...	...
25	CWE-77	109	6.38	0.75	519.95

## 6.2 Extending the Analysis

To further validate our metric, we extended our analysis beyond the Top 25 CWEs. As MITRE provides NVD counts and average CVSS scores only for the top 25 CWEs (plus an additional 15, for a total of 40), we needed to make this information available for all CWEs in order to compute Equation 9. After processing data from 2018 and 2019 (the same years used in the evaluation of the 2020 CWE Top 25), we found 27,437 CVEs and identified 179 CWEs, based on the data that was available in NVD as of the time of our experiments. As already noted in the previous subsection, the counts of CVEs in each CWE category that we identified differ slightly from MITRE’s official counts (as reported in Table 3 for the top 25 CWEs). These differences are due to the fact that MITRE did not provide a list of CVEs in each CWE category, but just their counts at the time the Top 25 ranking was computed. However, NVD is updated approximately every two hours irrespective of the CVE year, so a different number of CVEs may be returned when querying the database by year at different times. To verify that our own counts of CVEs in each CWE category would enable us to compute CWE scores consistent with those reported in Table 3, we evaluated the correlation between MITRE’s NVD counts and our own counts, and found the correlation to be about 90%.

## 6.3 Comparison for 2018-2019 CWEs

We conducted additional, separate experiments for the years 2018 and 2019. We calculated our overall scores for the 2018 CWEs and compared them against the CWE scores for the same year. The results, reported in Table 5, indicate a 90% correlation ( $R^2 = 0.81$ ), as shown in Figure 7. Note that the CWE scores that we computed for all the 2018 CWEs are not normalized, but this is irrelevant for the purpose of computing the correlation with our scores.

Table 5: Our ranking of 2018 weaknesses.

Rank	CWE ID	No. CVEs	Average Likelihood	Average Exp. Factor	Our Score
1	CWE-787	914	6.10	0.75	4,161.62
2	CWE-119	854	5.54	0.67	3,182.77
3	CWE-20	1,038	4.48	0.56	2,633.75
4	CWE-78	380	7.27	0.87	2,398.87
5	CWE-416	481	5.77	0.68	1,894.30
6	CWE-89	501	5.57	0.61	1,729.16
7	CWE-79	2,078	2.54	0.29	1,561.72
8	CWE-352	454	5.20	0.59	1,397.68
9	CWE-125	812	3.58	0.42	1,207.28
10	CWE-190	657	3.27	0.37	801.79

We repeated the same analysis for vulnerability data

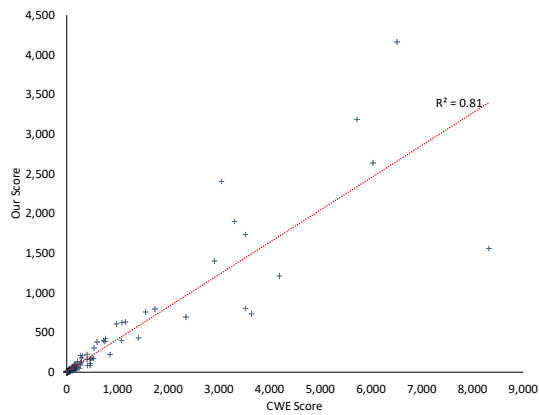


Figure 7: Correlation between MITRE's and our ranking of 2018 weaknesses.

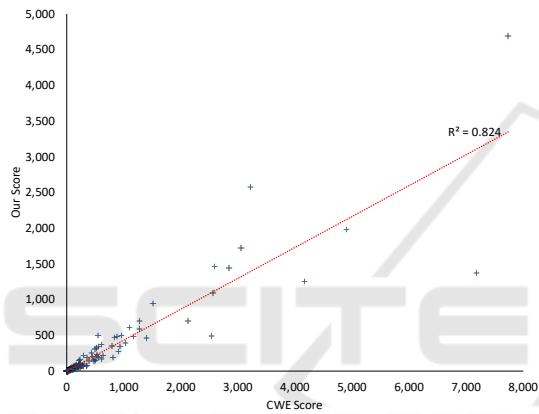


Figure 8: Correlation between MITRE's and our ranking of 2019 weaknesses.

from 2019. The results, reported in Table 6, indicate a 90.8% correlation ( $R^2 = 0.824$ ), as shown in Figure 8.

Table 6: Our ranking of 2019 weaknesses.

Rank	CWE ID	No. CVEs	Average Likelihood	Average Exp. Factor	Our Score
1	CWE-787	1,098	5.95	0.72	4,689.04
2	CWE-78	397	7.31	0.89	2,575.34
3	CWE-20	849	4.42	0.53	1,986.21
4	CWE-416	453	5.59	0.68	1,728.50
5	CWE-119	377	5.70	0.68	1,460.02
6	CWE-89	409	5.60	0.63	1,443.55
7	CWE-79	1,766	2.58	0.30	1,379.04
8	CWE-125	792	3.68	0.43	1,254.00
9	CWE-352	423	4.76	0.54	1,091.00
10	CWE-120	213	6.02	0.73	941.75

In conclusion, these results show that not only the proposed metrics are sound, as they are consistent with an established scoring system, but they also exhibit several important advantages:

- While CWE scores are computed at an aggregate

level, our equivalent score is calculated based on vulnerability-level metrics, allowing one to easily drill down the analysis at a finer level of granularity.

- Our system of metrics can separately model the likelihood and the impact of a vulnerability and offers a flexible approach to considering multiple variables and to adjusting their relative weights based on a specific application scenario and operational context.

### 6.4 Effect of Considering IDS Rules

The experiments described in the previous subsections have been run for  $\gamma = 0$  to ignore the effect of IDS rules on the likelihood, since our objective was to validate our metrics against CWE scores, which do not consider IDS rules. Therefore, using a practical example, we now show the effect of IDS rules on the computation of the likelihood. Consider the two vulnerabilities identified by CVD IDs CVE-2018-11776 and CVE-2018-12572. When the effect of IDS rules is ignored, CVE-2018-11776 appears to be slightly more likely. However, while CVE-2018-12572 has only one associated IDS rule in Snort, CVE-2018-11776 has 7 known IDS rules. Thus, when IDS rules are considered in the computation, the likelihood of CVE-2018-11776 gets drastically reduced, as reported in Table 7.

Table 7: Effect of IDS rules on likelihood.

CVE ID	No. IDS rules	Likelihood for $\gamma = 0$	Likelihood for $\gamma = 0.25$
CVE-2018-11776	7	0.9861	0.0141
CVE-2018-12572	1	0.8587	0.6680

Similarly, when the effect of deployed IDS rules is considered, the impact factor of a vulnerability with multiple deployed rules is significantly reduced to reflect the fact that such a vulnerability can be more easily mitigated.

### 6.5 Tuning of Parameters

The metrics presented in this paper involve several tunable parameters, namely  $\alpha$ ,  $\beta$ , and  $\gamma$  in Equation 10 and  $\delta$  in Equation 12. In Section 5, we explained how these parameters can be set to ignore the effect of a variable from the overall computation. More generally, these parameters can be used to weight the contribution of the different variables to the overall score. By adjusting these parameters, one can tune the results to specific applications and operational contexts. We plan to further investigate the subject of parameter

tuning as part of our future work. For the purpose of our analysis, we chose values of the parameters that maximized the correlation between our scores and CWE scores.

## 7 CONCLUSIONS

In this paper, we have introduced metrics to enable practical and effective application of graph-based configuration analytics and optimization. Our system of metrics builds upon literature on vulnerability graphs and vulnerability scoring, and can effectively complement systems like SCIBORG, a graph-based framework providing a fully automated pipeline to ingest information about a networked system, build a graph model of the system based on this information, and recommend configuration changes to optimize security while preserving functionality. In particular, we defined metrics to evaluate (i) the likelihood of exploiting a vulnerability, (ii) probability distributions over the edges of a vulnerability graph, and (iii) exposure factors of system components to vulnerabilities. Our approach builds upon standard vulnerability scoring systems, and we showed that the proposed metrics can be easily extended. We have evaluated our approach against the Common Weakness Scoring System (CWSS), showing a high degree of correlation between CWE scores and our metrics. As part of our future work, we plan to explore tuning of the parameters used in the likelihood and exposure factor equations and develop an overall metric to score and compare configurations.

## ACKNOWLEDGEMENTS

This work was funded by the US Department of Defense under the DARPA ConSec program. Any opinions expressed herein are those of the authors and do not necessarily reflect the views of the U.S. Department of Defense or any other agency of the U.S. Government.

## REFERENCES

- (2017). OWASP top 10 - 2017: The ten most critical web application security risks. Technical report, The OWASP Foundation.
- Albanese, M. and Jajodia, S. (2018). A graphical model to assess the impact of multi-step attacks. *Journal of Defense Modeling and Simulation*, 15(1):79–93. Selected by the Guest Editor, Alexander Kott, as an article of particular value.
- Albanese, M., Pugliese, A., and Subrahmanian, V. (2013). Fast activity detection: Indexing for temporal stochastic automaton-based activity models. *IEEE Transactions on Knowledge and Data Engineering*, 25(2):360–373.
- Ammann, P., Wijesekera, D., and Kaushik, S. (2002). Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, pages 217–224, Washington, DC, USA. ACM.
- Bahl, P., Barham, P., Black, R., Chandra, R., Goldszmidt, M., Isaacs, R., Kandula, S., Li, L., MacCormick, J., Maltz, D., Mortier, R., Wawrzoniak, M., and Zhang, M. (2006). Discovering dependencies for network management. In *Proceedings of the 5th ACM Workshop on Hot Topics in Networking (HotNets-V)*, pages 97–102, Irvine, CA, USA. ACM.
- Bopche, G. S., Rai, G. N., Denslin Brabin, D. R., and Mehtre, B. M. (2019). A proximity-based measure for quantifying the risk of vulnerabilities. In Thampi, S. M., Perez, G. M., Ko, R., and Rawat, D. B., editors, *Proceedings of the 7th International Symposium on Security in Computing and Communication (SSCC 2019)*, volume 1208 of *Communications in Computer and Information Science*, pages 41–59. Springer.
- Jajodia, S. and Albanese, M. (2017). *Theory and Models for Cyber Situation Awareness*, volume 10030 of *Lecture Notes in Computer Science*, chapter An Integrated Framework for Cyber Situation Awareness, pages 29–46. Springer.
- Jajodia, S., Noel, S., and O’Berry, B. (2005). *Managing Cyber Threats: Issues, Approaches, and Challenges*, volume 5 of *Massive Computing*, chapter Topological Analysis of Network Attack Vulnerability, pages 247–266. Springer.
- Kourtellis, N., De Francisci Morales, G., and Bonchi, F. (2015). Scalable online betweenness centrality in evolving graphs. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2494–2506.
- Leverage, D. J. and Byres, E. J. (2008). Estimating a system’s mean time-to-compromise. *IEEE Security & Privacy*, 6(1):52–60.
- Manadhata, P. K. and Wing, J. M. (2011). An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386.
- Mukherjee, P. and Mazumdar, C. (2018). Attack difficulty metric for assessment of network security. In *Proceedings of 13th International Conference on Availability, Reliability and Security (ARES 2018)*, Hamburg, Germany. ACM.
- Natrajan, A., Ning, P., Liu, Y., Jajodia, S., and Hutchinson, S. E. (2012). NSDMiner: Automated discovery of network service dependencies. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (IEEE INFOCOM 2012)*, pages 2507–2515, Orlando, FL, USA. IEEE.
- Soroush, H., Albanese, M., Asgari Mehrabadi, M., Iganibo, I., Mosko, M., Gao, J. H., Fritz, D. J., Rane, S., and Bier, E. (2020). SCIBORG: Secure configurations for the IoT based on optimization and reasoning on

- graphs. In *Proceedings of the 8th IEEE Conference on Communications and Network Security (CNS 2020)*. IEEE.
- Stuckman, J. and Purtilo, J. (2012). Comparing and applying attack surface metrics. In *Proceedings of the 4th International Workshop on Security Measurements and Metrics (MetriSec 2012)*, pages 3–6, Lund, Sweden. ACM.
- Venkatesan, S., Albanese, M., and Jajodia, S. (2015). Disrupting stealthy botnets through strategic placement of detectors. In *Proceedings of the 3rd IEEE Conference on Communications and Network Security (IEEE CNS 2015)*, pages 55–63, Florence, Italy. IEEE. Best Paper Runner-up Award.
- Wang, L., Zhang, Z., Li, W., Liu, Z., and Liu, H. (2019). An attack surface metric suitable for heterogeneous redundant system with the voting mechanism. In *Proceedings of the International Conference on Computer Information Science and Application Technology (CISAT 2018)*, volume 1168 of *Journal of Physics: Conference Series*, Daqing, China. IOP Publishing.
- Yoon, S., Cho, J.-H., Kim, D. S., Moore, T. J., Free-Nelson, F., and Lim, H. (2020). Attack graph-based moving target defense in software-defined networks. *IEEE Transactions on Network and Service Management*, 17(3):1653–1668.

