Measuring Similarity for Security Vulnerabilities

Ju An Wang, Linfeng Zhou, Minzhe Guo, Hao Wang, and Jairo Camargo
Southern Polytechnic State University
jwang@spsu.edu

Abstract
As the number of software vulnerabilities increases year by year, software vulnerability becomes a focusing point in information security. This paper proposes a vulnerability similarity measurement to compare different vulnerabilities according to a set of criteria. Our approach is based on the structural hierarchy of vulnerabilities, and the similarity is defined using established mathematical models. The National Vulnerability Database and the Ontology of Vulnerability Management provide the information necessary for the similarity calculation. The similarity measurement can be used in many areas of vulnerability management, such as vulnerability classification, mitigation, and patching.

1. Introduction
It is very difficult to develop software products completely perfect, as flaws and defects are always present in software products, giving attackers opportunities to exploit these vulnerabilities and attack a computer system. Vulnerabilities may result from software bugs, weak passwords, or system misconfigurations. CVE (Common Vulnerabilities and Exposures) [11], a standard dictionary with information about security vulnerabilities and exposures, includes a set of common identifiers that enable data exchange of security products and provide a baseline index point to evaluate coverage of tools and services [11]. As of April of 2009, there were 37,031 vulnerabilities listed in the National Vulnerability Database [13] and there are characteristics that are shared among several of those vulnerabilities. As a result, a similarity method for security vulnerability can speed up the process of vulnerability mitigation as similar vulnerabilities may require similar solutions. Furthermore, similarity methods can prioritize the remediation level of vulnerability; in other words, in order to take any action, a level of priority can be determined based on known data from other vulnerabilities. For example, a low priority level can be assigned to a newly found vulnerability if it is proved that it is similar to a known vulnerability that has a low impact. In this paper we propose a measurement of similarity between security vulnerabilities. With this metrics, information security professionals will be able to find vulnerabilities with similar characteristics and thus provide similar countermeasures.

Analyzing source code is an effective method to detect vulnerabilities and obtain vulnerability information; however, it is not feasible to reengineer a large number of software products, and sometimes, the source code is not available for this purpose. In our approach, we take the National Vulnerability Database [13] as our information source. The National Vulnerability Database stores the information of all vulnerabilities found so far in all kinds of applications and operating systems, providing full and reliable information for research in security vulnerability. Therefore, our plan is to retrieve this information and use the data to process our measurement, so that we can get all the vulnerability information about a system without having to scan the system’s source code. As mentioned before, the purpose of this paper is to measure similarities between security vulnerabilities through a hierarchical similarity algorithm, which uses information retrieved from NVD [13] through an ontology for vulnerability management [6] to get the similarities between vulnerabilities. Each vulnerability has several characteristics. Different characteristics have different algorithms to calculate their similarities.

The rest of the paper is organized as follows: Section 2 discusses OVM and related work; Section 3 provides our algorithm for vulnerability similarity; Section 4 provides a number of examples illustrating our approach; and Section 5 summarizes our approach and concludes the paper with a few further research topics.

2. Related Work
A vulnerability is a security flaw, which arises from computer system design, implementation, maintenance, and operation. For better vulnerability management and further knowledge discovery based on the existing efforts in vulnerability research, we have designed a vulnerability ontology OVM (ontology for vulnerability management) [6], which
can be populated with all existing vulnerabilities in NVD [13]. It supports research on reasoning about vulnerabilities, their characterization, and their impact on computing systems. Vendors and users can use our ontology as a support in vulnerability analysis, tool development and vulnerability management. For the purpose of this paper, OVM serves as the knowledge base and data source for measuring similarities among vulnerabilities.

The basis of our vulnerability ontology is built on the result of CVE [11] and its related protocols and standards including CWE (Common Weakness Enumeration) [12], CPE (Common Platform Enumeration) [10], and CAPEC (Common Attack Pattern Enumeration and Classification) [9]. The top-level concepts include Vulnerability, IT_Product, Attacker, Attack, Consequence, and Countermeasure. More specifically, a Vulnerability existing in an IT_Product can be exploited by an Attacker through an Attack with the objective of compromising the IT_Product and cause a Consequence. Countermeasures can be used to protect the IT_Product through mitigation of the Vulnerability. The OVM provides a set of utilities and a centralized repository where all participants can share and communicate effectively. This will speed up the development process, foster interoperability, improve correlation of test results and ease gathering of metrics.

Given a software product, we are able to retrieve all the vulnerability information for this product from our OVM. This provides an automated way to gather information about vulnerabilities related to a software product and its common attack patterns. Since this paper focuses on the similarity metrics, we will omit the details of how the vulnerability information associated with a software product is retrieved from OVM. Readers are referred to [6] for more details.

P. Ganesan et al. [3] studied the notion of similarity between objects and their similarity measurement. In An’s paper [1], his method focus on C Code security vulnerability detection. Based on Case-based Reasoning technology, the method of similarity matching between security characteristic of source code and the characteristic of security vulnerabilities is proposed in this paper, and the similarity degree determines if the source code has security vulnerabilities.

In Y. Wang’s paper [8], disease is compared to vulnerability for their similar characteristics. Many pathological perspectives are used in this paper to explain the definitions of vulnerabilities. A CR model of network vulnerability was presented to interpret cause-result of vulnerability. Similarly, this paper assessed the metric for network vulnerability with vulnerabilities’ characteristics such as exploitability and effect and the structure of vulnerability relation.

In Byers’ work [2], Vulnerability Cause Graphs is presented to analyze software vulnerabilities. The paper aims the cause of vulnerability to build a structured method for modeling software vulnerabilities.

Igure’s paper [4] is a survey for computer system security vulnerability. It lists many papers which have good ideas about security vulnerabilities’ classification and Taxonomies. From this paper, we can get the ideas that how researchers view the vulnerability taxonomy historically. Based on their research on the vulnerability taxonomy, we pick up several important vulnerability features to use in our measurement, such as some characteristics like Type, Impact (AI, CI, and II) and Exploitation (AV, AC, and Au).

3. Vulnerability Similarity Measurement

3.1 Overview

According to the vulnerability information from NVD [13], we can compare data between vulnerabilities, such as type, their presence in similar software, their impact, etc. If a security officer or a system administrator finds a very serious vulnerability that can harm a system and wants to find out all similar vulnerabilities in that system, our measurement could provide him that information. Moreover, our approach can determine exactly how similar they are by our similarity score.

Each vulnerability may have many key characteristics, and each characteristic may have a different impact on the vulnerability’s level of severity to the system; thus each characteristic’s weight should be different. In our measurement, we compute each characteristic’s similarity and then combine them to generate the final similarity score according to their influence weights. Also, since different characteristics may have different domains and methods to compute similarities, we adopted the hierarchical domain structure discussed in [3] in our measurement to calculate each characteristic’s similarity score.

3.2 The Overall Algorithm

There are eight major characteristics that have influence in vulnerability similarity measurement. These characteristics are Type, Product, Access Vector, Access Complexity, Authentication, Confidentiality Impact, Integrate Impact and Availability Impact. We retrieve the information we need about vulnerabilities from the National
Vulnerability Database [13] through the Ontology for Vulnerability Management (OVM) [6] and then we measure their similarity.

The vulnerability Type is one of the most important characteristics in our measurement. The type of vulnerability determines what attacks can exploit this vulnerability and how hackers may exploit the vulnerability. From the type, we can obtain much information about the vulnerability such as the possible types of products this vulnerability may exist in and the possible solutions to this vulnerability. The Product characteristic is another important factor to define vulnerability similarity. Usually, if two vulnerabilities exist in one product, there is some relationship or similarity between them. The Access Vector (AV), Access Complexity (AC) and Authentication (Au) are three factors taken from the CVSS base metrics [5]. They measure what methods, how complex, and how many authentications are required for the vulnerability to be exploited. Here we call these three factors as Exploitability factors. The Confidentiality Impact (CI), Integrate Impact (II), and Availability Impact (AI) are other three factors taken from CVSS base metrics [5] are Impact factors. They measure three different impacts on the system after the vulnerability has been successfully exploited. These six CVSS properties define the vulnerability to be exploited and how severe the vulnerability is after being exploited. Each factor has its own weight in our measurement according to its influence in the similarities between vulnerabilities. The overall formula is shown below:

\[ (1) \text{Sim}(V_1, V_2) = W_1 \cdot \text{SimType}(V_1, V_2) + W_2 \cdot \text{SimProduct}(V_1, V_2) + W_3 \cdot \text{SimAV}(V_1, V_2) + W_4 \cdot \text{SimAC}(V_1, V_2) + W_5 \cdot \text{SimAu}(V_1, V_2) + W_6 \cdot \text{SimCI}(V_1, V_2) + W_7 \cdot \text{SimII}(V_1, V_2) + W_8 \cdot \text{SimAI}(V_1, V_2) \]

Formula (1) covers all the factors we discussed previously. \( \text{Sim}(V_1, V_2) \) stands for the similarity between vulnerability \( V_1 \) and vulnerability \( V_2 \). The value of \( \text{Sim}(V_1, V_2) \) is a number between 0 and 10. The term \( \text{SimType}(V_1, V_2) \) refers to the vulnerability type similarity, measuring how similar the types of two vulnerabilities are. The term \( \text{SimProduct}(V_1, V_2) \) refers to the similarity between products that have both vulnerabilities \( V_1 \) and \( V_2 \). \( \text{SimAV}(V_1, V_2) \) is the similarity of the access vectors while exploiting the vulnerabilities. \( \text{SimAC}(V_1, V_2) \) is the similarity of access complexity while exploiting these vulnerabilities. \( \text{SimAu}(V_1, V_2) \) is the similarity of authentication while exploiting these vulnerabilities. \( \text{SimCI}(V_1, V_2) \) is the similarity of the confidential impact when the vulnerabilities are exploited. \( \text{SimII}(V_1, V_2) \) is the similarity of the integrate impact when the vulnerabilities are exploited. \( \text{SimAI}(V_1, V_2) \) is the similarity of the availability impact when these vulnerabilities are exploited. Each similarity characteristic such as \( \text{SimType}(V_1, V_2) \) has a value ranged from 0 to 1. \( \text{Sim}(V_1, V_2) \), the total score of similarity, has range from 0 to 10.

### 3.3 The Algorithms for the Similarity between Individual Characteristics

#### (1) Vulnerability Type Similarity

As we discussed previously, Type is an important characteristic of vulnerabilities. To compute the similarity between vulnerability types, we first built a hierarchical tree to calculate the type similarity. According to the CVE lists in NVD, there are 19 vulnerability types such as Buffer Error and SQL Injection. Among them, 14 vulnerability types are also in the CWE Research View [12]. Based on the structure of CWE Research View, we built a hierarchical tree for vulnerability types. In Figure 1, those nodes with shadow text are vulnerability types in NVD.

Based on the vulnerability type hierarchical tree, we use the following algorithm introduced in [3] to calculate the similarity:

\[ \text{SimType}(T_1, T_2) = \frac{2 \cdot \text{Depth}(LCA(T_1, T_2))}{\text{Depth}(T_1) + \text{Depth}(T_2)} \]

Here, \( T_1 \) and \( T_2 \) are vulnerability types for \( V_1 \) and \( V_2 \) respectively. The lowest common ancestor \( LCA(T_1, T_2) \) is defined as the node of greatest depth that is a common ancestor for both \( T_1 \) and \( T_2 \). \( \text{Depth}(T_1) \) and \( \text{Depth}(T_2) \) are defined as the depths of nodes \( T_1 \) and \( T_2 \) in the hierarchical tree [4].

Let us see an example for computing similarities between vulnerability types. Assume that the type of vulnerability \( V_1 \) is SQL Injection and the type of vulnerability \( V_2 \) is OS Command Injection. The depth of SQL Injection is 3, represented as \( \text{Depth}(T_1) = 3 \), and the depth of OS Command Injection is 4, represented as \( \text{Depth}(T_2) = 4 \). According to Figure 1, the deepest ancestor of the two vulnerability types is “Failure to Sanitize Data into a Different Plane (Injection)” with a depth of 2, shown as \( \text{Depth}(LCA(V_1, V_2)) = 2 \). Having defined this, we now can determine the similarity between the two vulnerability types as follows:
\[ SimType(T_1, T_2) = \frac{2 \cdot Depth(LCA(T_1, T_2))}{Depth(T_1) + Depth(T_2)} \]

\[ = \frac{2 \cdot 2}{3 + 4} = \frac{4}{7} \]

(2) Vulnerability Product Similarity

Software products have vulnerabilities, and security vulnerabilities in particular are often associated with certain software products or platforms. If two vulnerabilities exist in two similar products, then the chances of having similar vulnerabilities are high. In this way, since the Product similarity represents the similarity level of vulnerabilities hosted in platforms, we can compute the similarity between vulnerabilities on platforms based on the product hierarchical tree.

The process to calculate the product similarity is similar to that for vulnerability type similarity. We first build a product hierarchical tree from the Ontology for Vulnerability Management and then compute the similarity based on the similarity algorithm [6]. Due to space limitation, we only captured an excerpt of the product hierarchical tree from OVM in Figure 2.

![Vulnerability Type Hierarchy](image)
As shown in Figure 2, products instances, such as Sun_Java_System_Identity_Manager_6.0 and Sun_Java_System_Identity_Manager_7.1, are leaves on the hierarchy tree, while non-leaf nodes represent product categories.

One difference between product similarity and type of similarity is that a vulnerability can be present in several products despite that it only has one vulnerability type. In this case, the product similarity is not the similarity between two nodes in the product hierarchical tree; it is the similarity between two sets that are comprised of nodes in the product hierarchical tree. The original method to compute similarity between two sets was introduced in [3]. We use sets or collections to represent a group of products that are vulnerable to a specific vulnerability. As shown in Figure 2, vulnerable products are collections of leaves on the hierarchical tree. Therefore, the product collection similarity of two vulnerabilities can be turned into the similarity measurement of the two sets in the tree, and then apply and adjust the GCSM methods introduced in [3] to compute the hierarchical similarity of these two collections.

\[
\text{Sim Product}(C_1, C_2) = \frac{\vec{C}_1 \cdot \vec{C}_2}{\sqrt{\sum_{i=1}^{n} c_{1i} \cdot c_{1i}} \sqrt{\sum_{j=1}^{m} c_{2j} \cdot c_{2j}}}
\]

In this formula, \( n \) represents the number of members in collection \( C_1 \), and \( m \) the number of members in collection \( C_2 \). \( c_{1i} \) stands for the \( i \)th member in collection \( C_1 \) and \( c_{2j} \) for the \( j \)th member in collection \( C_2 \). For the purpose of hierarchical similarity measurement, we redefined the dot product of two collection members as shown below:

\[
\vec{c}_{i1} \cdot \vec{c}_{j2} = \frac{2 \cdot \text{depth}(LCA(c_{i1}, c_{j2}))}{\text{depth}(c_{i1}) + \text{depth}(c_{j2})}
\]

The lowest Common Ancestor \( LCA(c_{i1}, c_{j2}) \) is defined as the node of greatest depth that is a common ancestor for both \( c_{i1} \) and \( c_{j2} \). The \( \text{depth}(c_{i1}) \) is defined as the depth of node \( c_{i1} \) in the hierarchical tree [3].

Now let us use an example to better illustrate the similarity measurement between two sets of vulnerable products. Suppose collection \( C_1 \) is one set or collection of products vulnerable to vulnerability \( V1 \), and collection \( C_2 \) the set of products vulnerable to vulnerability \( V2 \).

\[
\begin{align*}
C_1 &= \{ \text{Sun: java_system_identify_manager 7.0, Sun: java_system_identify_manager 7.1}; \} \\
C_2 &= \{ \text{Sun: java_system_identify_manager 7.1, Sun: java_system_identify_manager 8.0}; \}
\end{align*}
\]

The product members in \( C_1 \) and \( C_2 \) are all the leaves in the product hierarchical tree as illustrated in Figure 2. To simplify the representation on the formula, we used \( c_{i1} (i = 1, 2) \) for members of collection \( C_1 \), and \( c_{j2} (j = 1, 2) \) for members of collection \( C_2 \). The similarity between these two collections is computed as follows:

\[
\begin{align*}
&n = 2, \\
&m = 2, \\
&\vec{c}_{i1} \cdot \vec{c}_{j2} = \frac{2 \cdot \text{depth}(LCA(c_{i1}, c_{j2}))}{\text{depth}(c_{i1}) + \text{depth}(c_{j2})} = 4/5, \\
&\ldots
\end{align*}
\]

\[
\text{Sim Product}(C_1, C_2) = \frac{\vec{C}_1 \cdot \vec{C}_2}{\sqrt{\sum_{i=1}^{n} c_{i1} \cdot c_{i1}} \sqrt{\sum_{j=1}^{m} c_{j2} \cdot c_{j2}}}
\]

\[
\begin{align*}
&= \sqrt{\sum_{i=1}^{n} c_{i1} \cdot c_{i1}} \sqrt{\sum_{j=1}^{m} c_{j2} \cdot c_{j2}}, \\
&= 0.884
\end{align*}
\]

The result shows that the similarity between vulnerability \( V1 \) and vulnerability \( V2 \) is 0.884.

(3) CVSS Factors Similarity

The next six factors in our measurement correspond to CVSS’ base metrics. The first three, Access Vector (AV), Access Complexity (AC) and Authentication (Au), are three factors from CVSS’ base score definition that measure what methods, how complex, and how many authentications are required for the vulnerability to be exploited. The other three factors, Confidentiality Impact (CI), Integrity Impact (II), and Availability Impact (AI),
are also factors from CVSS base metrics, but they measure three different impacts on the system after the vulnerability has been successfully exploited. These six CVSS properties define the vulnerability to be exploited and how severe the vulnerability is after being exploited. These factors are very important in the measurement of similarity because they measure the impact degree of a vulnerability.

Table 2 below shows the possible values for each factor used on the CVSS base metrics [5].

<table>
<thead>
<tr>
<th>CVSS Factors</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Vector(AV)</td>
<td>Local(L)</td>
<td>Adjacent</td>
<td>Network(N)</td>
</tr>
<tr>
<td></td>
<td>0.395</td>
<td>0.646</td>
<td>1.0</td>
</tr>
<tr>
<td>Access Complexity(AC)</td>
<td>High(H)</td>
<td>Medium(M)</td>
<td>Low(L)</td>
</tr>
<tr>
<td></td>
<td>0.35</td>
<td>0.61</td>
<td>0.71</td>
</tr>
<tr>
<td>Authentication(Au)</td>
<td>Multiple(M)</td>
<td>Single(S)</td>
<td>None(N)</td>
</tr>
<tr>
<td></td>
<td>0.45</td>
<td>0.56</td>
<td>0.704</td>
</tr>
<tr>
<td>Confidential Impact(CI)</td>
<td>None(N)</td>
<td>Partial(P)</td>
<td>Complete(C)</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.275</td>
<td>0.660</td>
</tr>
<tr>
<td>Integrity Impact(II)</td>
<td>None(N)</td>
<td>Partial(P)</td>
<td>Complete(C)</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.275</td>
<td>0.660</td>
</tr>
<tr>
<td>Availability Impact(AI)</td>
<td>None(N)</td>
<td>Partial(P)</td>
<td>Complete(C)</td>
</tr>
<tr>
<td></td>
<td>0.000</td>
<td>0.275</td>
<td>0.660</td>
</tr>
</tbody>
</table>

Table 2. Metrics in Base Metrics Group

From the table above we can see that each factor can have three different values, i.e. Access Complexity can have High, Medium or Low values. In addition, CVSS also gives each value a number between 0 and 1, e.g., the High value for Access Complexity is 0.35.

We define the distance between two values in the same factor as the absolute value between the two values assigned in CVSS (shown in Table 4). That is to say, the distance between the High value and Medium value of Access Complexity is |0.35 - 0.61| = 0.26. We also define the function MaxDistance to compute the maximum distance among the values of each factor:

$$\text{MaxDistance}(AC) = |\text{High} - \text{Low}| = 0.35 - 0.71| = 0.36$$

$$\text{MaxDis tan cte}(AV) = |\text{Network} - \text{Local}| = |0.0 - 0.395| = 0.605$$

$$\text{MaxDis tan cte}(Au) = |\text{Multiple} - \text{None}| = |0.45 - 0.704| = 0.254$$

$$\text{MaxDis tan cte}(CI) = |\text{Complete} - \text{None}| = |0.660 - 0| = 0.660$$

$$\text{MaxDis tan cte}(II) = |\text{Complete} - \text{None}| = |0.660 - 0| = 0.660$$

Next, we compute the similarity of each factor separately. Taking Access Complexity as an example, the similarity of Access Complexity in two vulnerabilities can be defined using the formula below:

$$\text{SimAC}(AC_1, AC_2) = 1 - \frac{|AC_1 - AC_2|}{\text{MaxDis tan cte}(AC)}$$

The inputs on SimAC are the values for Access Complexity in two vulnerabilities, AC1 and AC2. The output of SimAC is a number ranging from 0 to 1.

The other factor’s formulas are defined similarly as below:

$$\text{SimAV}(AV_1, AV_2) = 1 - \frac{|AV_1 - AV_2|}{\text{MaxDis tan cte}(AV)}$$

$$\text{SimAu}(Au_1, Au_2) = 1 - \frac{|Au_1 - Au_2|}{\text{MaxDis tan cte}(Au)}$$

$$\text{SimCI}(CI_1, CI_2) = 1 - \frac{|CI_1 - CI_2|}{\text{MaxDis tan cte}(CI)}$$

$$\text{SimII}(II_1, II_2) = 1 - \frac{|II_1 - II_2|}{\text{MaxDis tan cte}(II)}$$

$$\text{SimAI}(AI_1, AI_2) = 1 - \frac{|AI_1 - AI_2|}{\text{MaxDis tan cte}(AI)}$$

Here we give an example to illustrate our measurement. Assume that vulnerability V1’s Authentication value (Au1) is Multiple (0.45) and vulnerability V2’s Authentication value (Au2) is Single (0.56).

$$\text{MaxDis tan cte}(Au) = |\text{Multiple} - \text{None}| = 0.45 - 0.704 = 0.254$$

$$\text{SimAu}(Au_1, Au_2) = 1 - \frac{|Au_1 - Au_2|}{\text{MaxDis tan cte}(Au)} = 1 - \frac{0.45 - 0.56}{0.254} = 0.567$$

3.4. Weights for Individual Characteristics

Each characteristic has different weights because each factor has multiple influences on one vulnerability. For example, Vulnerability Type is more important than Authentication because Type is the basic property of a vulnerability, so the weight of Vulnerability Type is greater than the weight of Authentication.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Vulnerability Type</td>
<td>3.0</td>
</tr>
<tr>
<td>2. Product with vulnerability</td>
<td>2.6</td>
</tr>
<tr>
<td>3. Access Vector(AV)</td>
<td>1.0</td>
</tr>
<tr>
<td>4. Access Complexity(AC)</td>
<td>1.0</td>
</tr>
<tr>
<td>5. Authentication(Au)</td>
<td>1.0</td>
</tr>
<tr>
<td>6. Confidentiality Impact(CI)</td>
<td>0.8</td>
</tr>
<tr>
<td>7. Integrate Impact(II)</td>
<td>0.8</td>
</tr>
<tr>
<td>8. Availability Impact(AI)</td>
<td>0.8</td>
</tr>
<tr>
<td>Total</td>
<td>10.0</td>
</tr>
</tbody>
</table>

Table 3. Vulnerability Similarity Characteristics and Weight

From observation and research experience with NVD vulnerabilities, we differentiate the importance of each characteristic in terms of similarity measurement; that is why we recommend the weights shown in Table 3 for each characteristic. Those weights might be adjusted as more experiments are conducted in the future.

When we consider the weights for each characteristics, we may have Type, Product, Exploitability (AV, AC and Au) and Impact (CI, II,
At first, we consider these four have the same weight, which means each one takes up 25% of the total weight, but we did some adjusts for each weight. For one thing, according to the calculation of CVSS’s base score [5] which is used to present the severity degree of one vulnerability, the weights of Exploitability (AV, AC and Au) in the formula are higher than Impact (CI, AI and II). Roughly, $W_{\text{Exploitability}} : W_{\text{Impact}} = 5 : 4$. Therefore, in our method, the weight of AV, AC and Au is 1, and the weight of CI, AI and II is 0.8. For another, as we know, the type of a vulnerability are very important identification as we can figure out what kind of vulnerable code causes this vulnerability and even what part of software system this vulnerability may exist according to its type. When we want to solve a vulnerability in software system, the first thing we want to know is what kind of vulnerability it is. So here we give Type weight 3. Product is also a very important characteristic but not as type. Here we give weight 1.6 so that the total weight will be 10.

### 4. Examples

In this section, we will give two examples to illustrate our method. The first example is similarity score comparisons among three vulnerabilities, and the second one is a comparison between traditional measurement and our hierarchical measurement.

#### 4.1. Hierarchical Similarity Example

The following example presents a comparison for similarity scores among three vulnerabilities. We choose three vulnerabilities from NVD and compared each one with the others using our similarity measurement. The three vulnerabilities were CVE-2009-1081, CVE-2009-1083 and CVE-2009-5114. Table 4, 5 and 6 shows the information of these three vulnerabilities as retrieved from NVD.

**Table 4. Information of CVE-2009-1081**

<table>
<thead>
<tr>
<th>CVE</th>
<th>Type</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-1081(V1)</td>
<td>XSS</td>
<td>Sun: java_system_identify_manager 7.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun: java_system_identify_manager 7.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun: java_system_identify_manager 7.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun: java_system_identify_manager 8.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Vector</th>
<th>Network</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Access Complexity</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>None</td>
</tr>
<tr>
<td>Confidentiality Impact</td>
<td>None</td>
</tr>
<tr>
<td>Integrate Impact</td>
<td>Partial</td>
</tr>
<tr>
<td>Availability</td>
<td>None</td>
</tr>
</tbody>
</table>

**Table 5. Information of CVE-2009-1083**

<table>
<thead>
<tr>
<th>CVE</th>
<th>Type</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2009-1083(V2)</td>
<td>Code Injection</td>
<td>Sun: java_system_identify_manager 7.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun: java_system_identify_manager 7.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun: java_system_identify_manager 7.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun: java_system_identify_manager 8.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Vector</th>
<th>Network</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Access Complexity</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>None</td>
</tr>
<tr>
<td>Confidentiality Impact</td>
<td>None</td>
</tr>
<tr>
<td>Integrate Impact</td>
<td>Partial</td>
</tr>
<tr>
<td>Availability</td>
<td>None</td>
</tr>
</tbody>
</table>

**Table 6. Information of CVE-2008-5114**

<table>
<thead>
<tr>
<th>CVE</th>
<th>Type</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2008-5114(V3)</td>
<td>XSS</td>
<td>cpe/a: sun: java_system_identity_manager:6.0:sp1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cpe/a: sun: java_system_identity_manager:6.0:sp2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cpe/a: sun: java_system_identity_manager:6.0:sp3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cpe/a: sun: java_system_identity_manager:6.0:sp4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cpe/a: sun: java_system_identity_manager:6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cpe/a: sun: java_system_identity_manager:7.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cpe/a: sun: java_system_identity_manager:7.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Vector</th>
<th>Network</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Access Complexity</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>None</td>
</tr>
<tr>
<td>Confidentiality Impact</td>
<td>None</td>
</tr>
<tr>
<td>Integrate Impact</td>
<td>Partial</td>
</tr>
<tr>
<td>Availability</td>
<td>None</td>
</tr>
</tbody>
</table>

We calculate the similarity on each characteristic first and then combine them together according to their weights.

**Type Similarity:**

$V1$’s type is XSS, $V2$’s type is Code injection and $V3$’s type is XSS. According to the vulnerability hierarchical tree in Figure 1, $\text{Depth}(V1) = 3$, and $\text{Depth}(V2) = 3$, and thus, $LCA(V1, V2) = 2$. Using the equation in Section 3.2 to calculate the type similarity, we have obtained the following result:

$$ \text{SimType}(V1, V2) = \frac{2 \cdot 2}{3 + 3} = 0.667 $$

Using the same method to compute the Type Similarity between $V1$ and $V3$ and between $V2$ and $V3$, we obtain:

$$ \text{SimType}(V1, V3) = 1 $$

$$ \text{SimType}(V2, V3) = 0.667 $$

**Product Similarity**

In Table 7 we listed the platforms affected by $V1$, $V2$ and $V3$ respectively.
Table 7. Product Information of Three Vulnerabilities

<table>
<thead>
<tr>
<th>Products</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun: java_system_identify_manager 7.0</td>
<td>Sun: java_system_identify_manager 7.1</td>
<td>Sun: java_system_identify_manager 7.11</td>
<td>Sun: java_system_identify_manager 8.0</td>
</tr>
<tr>
<td></td>
<td>Sun: java_system_identify_manager 7.0</td>
<td>Sun: java_system_identify_manager 7.1</td>
<td>Sun: java_system_identify_manager 7.11</td>
</tr>
<tr>
<td></td>
<td>Sun: java_system_identify_manager 7.0</td>
<td>Sun: java_system_identify_manager 7.1</td>
<td>Sun: java_system_identify_manager 7.11</td>
</tr>
</tbody>
</table>

Table 8. CVSS factors Information for Two Vulnerabilities

<table>
<thead>
<tr>
<th>CVSS Factor</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Vector (AV)</td>
<td>0.646</td>
<td>0.646</td>
<td>0.646</td>
</tr>
<tr>
<td>Access Complexity (AC)</td>
<td>0.61</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>Authentication (AC)</td>
<td>0.704</td>
<td>0.704</td>
<td>0.704</td>
</tr>
<tr>
<td>Confidentiality Impact (CI)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Integrity Impact (II)</td>
<td>0.275</td>
<td>0.275</td>
<td>0.275</td>
</tr>
<tr>
<td>Availability Impact (AI)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Overall Similarity

The similarity between CVE-2009-1081 (V1) and CVE-2009-1083 (V2) was obtained as follows:

\[
Sim(V1, V2) = \sum_{i=1}^{8} \alpha_i \cdot \prod_{j=1}^{8} \prod_{k=1}^{8} \text{Sim}_{ij}^{jk}
\]

The similarity between CVE-2009-1083 (V1) and CVE-2008-5114 (V3) was

\[
Sim(V1, V3) = \sum_{i=1}^{8} \alpha_i \cdot \prod_{j=1}^{8} \prod_{k=1}^{8} \text{Sim}_{ij}^{jk}
\]

And the similarity between CVE-2009-1083 (V2) and CVE-2008-5114 (V3) was

\[
Sim(V2, V3) = \sum_{i=1}^{8} \alpha_i \cdot \prod_{j=1}^{8} \prod_{k=1}^{8} \text{Sim}_{ij}^{jk}
\]

We tried to find out a method to validate our measurement for vulnerability similarity, but so far, we did not find any paper that has the same purpose with ours, so we cannot validate our measurement using previous work. Now what we can do is to validate our result according to security experts’ examination. From the calculation above we can figure out that V1 and V3 are the most similar. The second one is V1 and V2. The least similar one is V2 and V3. From our observation of V1, V2 and V3’s information, V1 and V3 have all the characteristics same except a few products and AC, in this case, the similarity between V1 and V3 should be very high, which matches the result of Sim(V1, V3)=9.152. V1 and V2 have only Type and AC different, because the weight of Type is higher than weight of Product, so the Sim(V1, V2) is not so high as Sim(V1, V3), but this score is still be considered high. The similarity between V2 and V3 is not high because they have
they have Type and Products different. All of the similarity scores match our intuition about vulnerability similarity, which shows that our measurement is right.

4.2. Comparison between hierarchical similarity measurement and traditional similarity measurement

Traditional similarity measurement uses a flat domain similarity method to compute each characteristic. Here we briefly introduce the method of traditional similarity measurement because it is easier to understand based on the contents we have introduced above.

**Type Similarity:**
If two vulnerabilities’ types are different then SimType(T1, T2) = 0, if they are same, then SimType(T1, T2) = 1;

**Product Similarity:**
We give an example: Suppose V1 and V2 have the following product sets P1 and P2 respectively:

P1=

- java_system_identify_manager 7.0
- java_system_identify_manager 7.1

P2=

- java_system_identify_manager 6.0
- java_system_identify_manager 7.1

Then we calculate their product similarity as:

\[
\text{SimProduct}(P1, P2) = \frac{2 \cdot \text{NumberShared}(P1, P2)}{\text{Number}(P1) + \text{Number}(P2)}
\]

where NumberShared(P1, P2) means the shared number of elements on P1 and P2. In this example NumberShared(P1, P2) is 1, because they share java_system_identify_manager 7.1. Number(P1) means the elements of P1, which is 2 in this example. So in this example SimProduct = 2*1/2+2 = ½.

**CVSS Similarity:**
Taking AV for example, if the AV’s of two vulnerabilities are different, then SimAV(AV1, AV2) = 0; if they are the same, then SimAV(AV1, AV2) = 1. Similar arguments apply to the other five factors.

**Overall Similarity**
First, we used traditional similarity measurement to compute similarities between V1, V2, and V3. V1, V2 and V3 are three vulnerabilities used in Section 4.1

\[
\begin{align*}
\text{SimType}(V1, V2) &= 0 \\
\text{SimType}(V1, V3) &= 1 \\
\text{SimType}(V2, V3) &= 0 \\
\text{SimProduct}(V1, V2) &= 1 \\
\text{SimProduct}(V1, V3) &= 0.18 \\
\text{SimProduct}(V2, V3) &= 0.18 \\
\text{SimAV}(V1, V2) &= 1 \\
\text{SimAV}(V1, V3) &= 1 \\
\text{SimAV}(V2, V3) &= 1 \\
\text{SimAV}(V1, V2) &= 0 \\
\text{SimAV}(V1, V3) &= 0 \\
\text{SimAV}(V2, V3) &= 1 \\
\text{SimAv}(V1, V2) &= 1 \\
\text{SimAv}(V1, V3) &= 1 \\
\text{SimAv}(V2, V3) &= 1 \\
\text{SimCI}(V1, V2) &= 1 \\
\text{SimCI}(V1, V3) &= 1 \\
\text{SimCI}(V2, V3) &= 1 \\
\text{SimII}(V1, V2) &= 1 \\
\text{SimII}(V1, V3) &= 1 \\
\text{SimII}(V2, V3) &= 1 \\
\text{SimAI}(V1, V2) &= 1 \\
\text{SimAI}(V1, V3) &= 1 \\
\text{SimAI}(V2, V3) &= 1 \\
\end{align*}
\]

According to the calculation above we can get the results below:

\[
\begin{align*}
\text{Sim}(V1, V2) &= 6.0 \\
\text{Sim}(V1, V3) &= 7.68 \\
\text{Sim}(V2, V3) &= 5.68 \\
\end{align*}
\]

The comparison between these two measurements is shown on Table 9 below. It turns out that traditional measurement focuses on simply counting the number of common items in two collections, whereas the hierarchical measurement of vulnerability similarity incorporates information from the hierarchical domain structure, therefore agreeing with our intuition idea of similarity much more closely.

Traditional Score is not as good as hierarchical measurement in our intuition. In the table 9, we can find that Sim(V1, V2) of Traditional measure is 4.4. From the information of vulnerability 1 and vulnerability 2 we can see that these two
vulnerabilities have the same Product, AV, Au, CI, II and AI. The different parts are Type and AC. Most of the two vulnerabilities’ information is the same, so from our intuition, we feel the similarity score between these two should be very high. But the traditional measurement gives only 6.0, which is not as good as hierarchical measurement score 8.732.

<table>
<thead>
<tr>
<th>Similarity</th>
<th>Hierarchical Measurement</th>
<th>Traditional Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sim(V1, V2)</td>
<td>8.732</td>
<td>6.0</td>
</tr>
<tr>
<td>Sim(V1, V3)</td>
<td>9.152</td>
<td>7.688</td>
</tr>
<tr>
<td>Sim(V2, V3)</td>
<td>7.808</td>
<td>5.688</td>
</tr>
</tbody>
</table>

Table 9 Hierarchical and Traditional Scores

5. Discussion and Conclusion

As the number of security vulnerabilities increases, it is important to classify them with a systematic scheme and evaluate their similarities. This paper discussed a set of similarity algorithms for comparing security vulnerabilities. Our approach is based on the structural hierarchy of vulnerabilities, and the similarity is defined using established mathematical models. There are some issues to be solved in the future work. For example, the data such as the weight of each characteristic used in our measurement are still on experiment. We need more information and real-world examples to validate our calculation and to reach a refined formula for the metrics. As mentioned above, our measurement highly depends on the organization of the hierarchy structure from vulnerability characteristics, so building the hierarchy structure for vulnerability is the first step to our measurement. So far we have preliminary hierarchy for types and products and they will be improved in our future work.

The following research topics merit further effort: First, an automated tool similar to the OSAT tool in [7] will help calculate the similarity between two vulnerabilities. One solution is to integrate the similarity calculation into OSAT with the same user interface, so users could have a similar experience with the automated tool. Second, more sophisticated similarity models were presented in [3], and it is worth further studies in their suitability in the calculation of vulnerability similarity. Finally, we believe that the approaches discussed in this paper apply to similarity research for other vulnerability related information, for instance, similarity of security threats and security countermeasures, thus it is interesting to see how similarity measurement will be defined for security weakness, attacks, risks, and countermeasures.

Acknowledgment

The authors would like to express their gratitude to anonymous reviewers of this paper for their thoughtful comments and suggestions.

7. References


