VRank: A Context-Aware Approach to Vulnerability Scoring and Ranking in SOA

Jianchun Jiang\textsuperscript{8}, Liping Ding\textsuperscript{8}, Ennan Zhai\textsuperscript{5}, Ting Yu\textsuperscript{4}

\textsuperscript{8}Institute of Software, Chinese Academy of Sciences
\textsuperscript{4}North Carolina State University
Email: \{jianchun, liping, ennan\}@nfs.iscas.ac.cn, yu@csc.ncsu.edu

\textbf{Abstract}—With the rapid adoption of the concepts of Service Oriented Architecture (SOA), sophisticated business processes and tasks are increasingly realized through composing distributed software components offered by different providers. Though such practices offer advantages in terms of cost-effectiveness and flexibility, those components are not immune to vulnerabilities. It is therefore important for the administrator of some composed service to evaluate the threats of such vulnerabilities accordingly within limited available information. Since almost all the existing efforts (e.g., CVSS) fail to consider specific context-aware information which is the specific character of SOA, they could not be adopted into SOA for scoring vulnerabilities.

In this paper, we present VRank, a novel framework for the scoring and ranking of vulnerabilities in SOA. Different from existing efforts, for a given vulnerability, VRank not only considers its intrinsic properties (e.g., exploitability), but also takes into account the contexts of the services having this vulnerability, e.g., what roles they play in the composed service and how critical it is to the security objective of the service. The resulting scoring and ranking of vulnerabilities are thus highly relevant and meaningful to the composed service. We present the detailed design of VRank, and compare it with CVSS. Our experiments indicate VRank is able to provide much more useful ranking lists of vulnerabilities for complex composed services.

\textbf{Keywords}—Vulnerabilities, Service-Oriented Architecture, Context, Ranking and Scoring.

\section*{I. INTRODUCTION}

\textbf{A. Background and Motivation}

With the rapid adoption of the concepts of Service Oriented Architecture (SOA), we recently have witnessed an increasing trend for organizations and enterprises to implement sophisticated business processes by composing distributed service components offered by different software providers (either private or public), which offers significant advantages in terms of cost-effectiveness, extensibility and flexibility. On the other hand, some recent studies (e.g., [1]) indicated that such a shift of paradigm inevitably introduces some new security risks. Specifically, since various services in SOA are often built on diverse hardware and software platforms and offered by different providers, the types of vulnerabilities faced by composed services are also much more diverse than the ones in traditional software development [2]. To deal with this problem, it is critical to first understand how and to what extend each type of vulnerability affects the overall security of a business process. We use business process to denote a group of related services and the dependence relationships between them. For example, as shown in Figure 1, it shows us a typical business process, which is realized through composed services (i.e., \(S_1, S_2, S_3, S_4, S_5, S_n, S_m\)). Unfortunately, some of the services contain one or more vulnerabilities. It would be very helpful if the administrator of the business process could understand which of the four types of vulnerabilities has the biggest impact on its business, thereby enabling the one to patch the vulnerabilities according to their importance.

Long-lasting efforts, in fact, have been devoted to the disclosure and severity ranking of service vulnerabilities. There are thousands of new service vulnerabilities that are identified and publicized [3] each year. Moreover, public vulnerability databases are maintained, which contain classification of vulnerabilities, description of the nature of their threats, as well as some severity scores based on the feedback of security experts. For instance, FIRST’s Common Vulnerability Scoring System (CVSS) [4] seems to emerge as the de facto standard in the community, which was designed based on expert knowledge and feedbacks.

However, existing vulnerability systems are only able to rank vulnerabilities by a general way, i.e., how severe (or critical) they are, considering the scale that they exist in all the services deployed globally\textsuperscript{1}. Applying it directly to a particular business process (say, in SOA) may not always be meaningful.

\textsuperscript{1}Indeed, this should be one of the factors considered by security experts when they rank the impacts of vulnerabilities.
or instructive. For example, a denial of service vulnerability in MS SQL server might have a high severity score in CVSS; however, for a particular business process in SOA, its service components running MS SQL server might only be used for exception event logging, which is rarely invoked. Therefore, this vulnerability’s impacts on the availability of the whole process would not be as significant as suggested by its ranking score in CVSS. Moreover, another disadvantages of current vulnerability scoring systems are from the specific characters of composed services themselves. For two different business processes in SOA, it is entirely possible that the contexts of the business processes (i.e., the dependence relationships of services in the two businesses) are totally different; thus, a “small” vulnerability of an important service will probably much badly influence the whole business process. For instance, some legacy systems may not be widely used anymore in most of today’s IT services; thus, vulnerabilities specific to those systems might not be considered very critical in a general vulnerability database; nevertheless, if a business process has to rely on such legacy systems for its core functionality, their vulnerabilities will be ranked high and handled with high priority.

Almost all the current efforts in this field fail to consider any context-aware factors in their design. Here, we use context to denote the explicit and implicit dependence relationships of services in any given business process. Although the newest version of CVSS system does allow to input context information when scoring a given vulnerability, including potential loss due to a vulnerability, percentage of vulnerable systems as well as temporal factors (i.e., how a vulnerability changes over time), the scoring of such factors is still general, and largely relies on the experience of the administrator of an organization. There are still no systematic guidelines providing any specification on how to appropriately set these factors.

Based on the above analysis, for the administrator of some business process, it is obvious very desirable to have a tool, which not only considers the properties of vulnerabilities, but more importantly takes into account the context-aware information of the business process, and provides a severity score and ranking of each vulnerability to serve as a guideline regarding which vulnerabilities are the most urgent and critical ones for that particular business process, thereby enabling appropriate resources to be allocated to deal with them accordingly.

B. Our Approach and Contributions

In order to satisfy the above requirement, this paper presents VRank, a novel framework for the ranking of vulnerabilities of business processes in SOA. VRank is able to leverage existing vulnerability databases (e.g., CVE [5]), and explicitly considers the specific and detailed context of a business process. It scores some vulnerability $v$ by combining three factors: 1) the importance of the services with vulnerability $v$ of the given business process; 2) the threat level of $v$ with respect to a given security objective (e.g., confidentiality, integrity and availability); and 3) the exploitability of $v$, which reflects the probability that $v$ can be exploited from a certain service $s$.

To the best of our knowledge, VRank is the first effort toward automatic and context-aware scoring and ranking of vulnerabilities for composed services (i.e., business process) in SOA. Specifically, our contributions are listed as follows:

- Instead of advocating a fixed strategy to score and rank vulnerabilities, we design an extensible framework (i.e., VRank) for the ranking of vulnerabilities for business processes realized through service compositions in SOA. We identify the major components for context-aware vulnerability ranking, including vulnerability databases, service importance evaluation functions, and scoring functions, and define clear interaction interfaces between these components. The security administrators of a business process can then materialize these components to have a vulnerability ranking system that is the most suitable for their own business practice.

- We show a concrete materialization of the above framework. We leverage existing system such as CVSS and graph ranking algorithms, and also design novel vulnerability scoring functions that nicely integrate the properties of individual vulnerabilities with service importance in the context of specific business processes. The generated scores could be intuitively interpreted, which is lacking in current existing vulnerability ranking systems.

- We demonstrate the effectiveness of VRank based on three case studies, with various business process topologies and complexities in SOA. Our case studies indicate that, compared with CVSS, the vulnerability ranking result returned by VRank is much more meaningful and instructive for security administrators to better understand the actual threats of various vulnerabilities to their own business processes.

Note that we do not want to use VRank to replace general vulnerability databases such as CVSS and CVE [5]. Instead, the goal of VRank is to leverage as many as possible existing efforts on general vulnerability ranking and integrate them with the context of specific business processes, and provide more precise and specific vulnerability assessment for SOA.

II. DESIGN OF VRANK FRAMEWORK

To be context-aware, we have to bring domain knowledge of business processes into the ranking and scoring of vulnerabilities. Such domain knowledge includes the structure and semantics of a business process, as well as the preferences of its owner and administrator. After all, they are the people who know the business process best and exactly what they want out of it. Therefore, one important design principle of VRank is flexibility in configuration. It should allow administrators to easily configure VRank, and provide their own input to make it fit best for their business purposes. In the design of VRank, instead of advocating a fixed ranking strategy or algorithm, it is more important to identify the key components of a context-aware vulnerability ranking system, and clearly define the interfaces for their interactions.
Fig. 2: The structure and assignment procedure of VRank. Step 1: VRank’s users enter two key information, i.e., security requirement and context; Step 2: after receiving business context and security requirement, VRank computes scores for all the vulnerabilities by combining three factors extracted from three components, i.e., importance algorithm, VE_DB and TL_DB. Here, VE_DB denotes the database storing vulnerabilities’ exploitability, and TL_DB is the database with vulnerabilities’ threat levels for various security requirements. In the final step, a ranking list is produced by our framework.

As mentioned early, VRank is a framework whose components can be flexibly configured by security administrators to obtain a vulnerability ranking system that is most suitable to their business contexts. We call this process a materialization.
of VRank. Next, we show a materialization of VRank by implementing each component described above, including the design of two scoring functions.

III. A Materialization of VRank

This section aims to present detailed materialization of VRank. At the end of this section, we will discuss main differences between VRank and existing efforts.

A. Dependency Graph

A dependency graph captures the structure of the given composed business process, in particular the dependency relation between service components with regard to a security requirement. Formally, in our implementation, a dependency graph is modeled as a directly graph \((S, E)\), where each \(s \in S\) is a service in SOA, and each \((s_1, s_2) \in E\) represents a dependency, i.e., service \(s_2\) depends on service \(s_1\). As mentioned early, depending on the security requirement in question, an edge in \(E\) may represent a control flow dependency, an information flow dependency, or some other relationship between service components.

Though the above definition of dependency graph is simple, it can capture many security relevant properties between service components. It also serves as the basis for many possible extensions by introducing more semantic features. For example, besides simple edges, we may further define the relation between edges out of the same service node, such as branching and parallel execution. We may also introduce edge properties, such as a weight to represent the strength or the importance of a dependency. For simplicity, in the rest of the discussion, we stick with the above simple definition of dependency graphs.

B. Service Importance Function

There are many possible ways to estimate a service’s importance in a business process. For example, we may check whether (and how many times) a service is at a critical path of a graph. Given the simple definition of dependency graph above, we adopt the PageRank algorithm [9] to be the service importance function in our materialization. Intuitively, if many services point to a particular service \(v\), \(v\) is likely to play an important role in the whole business processes. Similarly for the case when \(v\) points to many other services (like a hub). This is in line with the essential idea behind PageRank. Of course, other link analysis algorithms, such as HITS [10], can also be adopted. In general, we believe that PageRank is one representative service importance function that is expected to be useful for many business process contexts.

When the dependency graph has richer semantic features, the importance function can be enhanced accordingly. For example, for a dependency graph with weighted edges, the probability of moving from one node to another following an edge in PageRank could be adjusted proportionally to the weight of that edge. As another example, if we aim to consider branching (following one edge or another out of the same node, similar to an OR relation) and parallel execution (following both edges out of the same node, similar to an AND relation), then the AssetRank algorithm [11] can be adopted instead.

C. Exploitability and Threat Level Databases

Exploitability. The exploitability database is obtained directly from CVSS, which is similarly done in other vulnerability assessment work (e.g., [12]). Actually, leveraging existing efforts to enable vulnerabilities ranking to be more precise is a reasonable scheme, because: 1) obtaining exploitability from some existing systems is very convenient and swift; and 2) some existing vulnerability databases have rich experience in the exploitability of vulnerabilities. Therefore, in the design of current VRank, we extract exploitability from some existing representative vulnerability databases such as CVSS.

Threat level. The level of threat database is partially derived from CVE. Here, we map CVE’s qualitative description of threat levels (none, partial, complete) to a numerical scale 1–5. Notice that the threat levels of some vulnerability (e.g., \(v\)) are based on the specific security objective (e.g., confidentiality, integrity and availability) or requirement. For instance, for a given vulnerability \(v\), its threat level in integrity might be 4 and the one in availability could be 1; thus, we can easily understand protecting the integrity of \(v\) should be more important than availability. Of course, this also needs to depend the exploitability of \(v\).

Queries. Queries to the exploitability and level of threat databases can be viewed as the applications of two functions: 1) \(E(v, s)\), which takes as input a service \(s\) and a vulnerability \(v\), and returns a numerical value to indicate \(v\)’s exploitability in \(s\); and 2) \(F(v, P)\), which takes as input a vulnerability \(v\) and a security objective \(P\), and returns \(v\)’s level of threat to \(P\).

D. Vulnerability Scoring Functions

We present the design of two scoring functions as well as the rationale behind the design. They both take into account the importance of services and vulnerability exploitability and level of threats, but differ in how the three types of factors are combined.

The first scoring function is defined as follows.

\[
R(v) = \alpha \cdot \sum_{i=1}^{\vert S_v \vert} (E(v, s_i) \cdot \text{Rank}(s_i)) + \beta \cdot F(v, P) \cdot \sum_{i=1}^{\vert S_v \vert} \text{Rank}(s_i)
\]

(1)

Where:

- \(S_v\): The set of services which contain the vulnerability \(v\).
- \(s_i\): The element of the set \(S_v\), i.e., the service which has the vulnerability \(v\).
- \(E(v, s_i)\): The function that returns the exploitability of \(v\) in service \(s_i\) (by querying VE_DB).
- \(\text{Rank}(s_i)\): The importance of service \(s_i\), i.e., score of service \(s_i\) computed by the service importance algorithm.
- \(P\): The security objective.
• $F(v, P)$: The function that returns the level of threat of vulnerability $v$ with respect to security requirement $P$ (by querying TL_DB).
• $\alpha$ and $\beta$: two adjustable parameters whose ranges are $[0, 1]$. We will mention the evaluations of $\alpha$ and $\beta$ later.

Here is the intuition behind the design of the above scoring function:

1) The significance of a vulnerability $v$’s exploitability and level of threat to the whole business process should depend on the roles that those services containing $v$ play in the given process; therefore, the importance measurements of services returned by the service importance algorithm are used as weights when taking $v$’s exploitability and level of threat into its severity score.

2) Exploitability and threat levels are two major properties for a vulnerability. For security administrators, it is desirable to be able to adjust their contributions to the severity of a vulnerability based on their own preferences. Therefore, in Equation (1), we first separate the effects of each property intentionally, and then combine them through two adjustable parameters $\alpha$ and $\beta$. The choices of these two parameters reflect the tradeoff of the two properties, given the domain knowledge of a security administrator.

Note that in real world scenario, VRank will not fix the values of the two adjustable parameters $\alpha$ and $\beta$, because it should depend on some actual conditions such as the requirements of the enterprise and the rationale behind certain security policies. Security administrators can manage the values of parameters $\alpha$ and $\beta$ according to their experiences and concrete security policies.

**A deeper discussion about the design.** In practice, the computation of importance of services may needs to take into account the weights of dependencies between services, i.e., looking dependencies differently rather than equally, and this will embody the context of business process more than the current one. For this case, VRank allows administrators to adapt the computation of $\text{Rank}(s_i)$ according to concrete requirements, since $\text{Rank}(s_i)$ is actually a general importance algorithm which can be adapted based on different cases. In addition, as mentioned earlier, though current VRank computes the importance of services with PageRank algorithm; however, if more specific relationships, e.g., “OR” and “AND”, between services need to be considered, AssetRank [11] can be used as importance algorithm instead of PageRank; similarly, if some services in a target business context need to be assigned reputation (or other ranking) values, we could leverage TrustRank [13] to score and differentiate each service. Since current vulnerability scoring systems such as CVSS and USCERT do not support such flexibility, VRank’s deployability and suitability are much better than existing efforts.

**Example.** We present an example shown in Figure 3 to illustrate equation (1). In this example, we build nine services ($s_1$ to $s_9$) and some of them contain several vulnerabilities ($v_1$ to $v_3$). A security administrator Alice needs to rank the three vulnerabilities in the context of the composed business process. She inputs the dependency graph $G$ of services (shown in Figure 3) into VRank; meanwhile, she sets the security requirement for her ranking to be “integrity”. For simplicity, we assume the exploitability of vulnerabilities in different services to be 1.0, i.e., $E(v_1, s_1) = E(v_1, s_4) = E(v_2, s_4) = E(v_2, s_7) = E(v_3, s_6) = E(v_3, s_9) = 1$. For the threat levels of different vulnerabilities, (the security requirement is integrity, i.e., $P$ = “integrity”), we assume $F(v_1, P) = 1$, $F(v_2, P) = 2$, and $F(v_3, P) = 3$. VRank first obtains $\text{Rank}(s_i)$ of all the services by using the PageRank algorithm over $G$, thus getting $\text{Rank}(s_1) = 0.15$, $\text{Rank}(s_2) = 0.21$, $\text{Rank}(s_3) = 0.21$, $\text{Rank}(s_4) = 0.24$, $\text{Rank}(s_5) = 0.24$, $\text{Rank}(s_6) = 0.33$, $\text{Rank}(s_7) = 0.35$, $\text{Rank}(s_8) = 0.5$ and $\text{Rank}(s_9) = 1.01$. Then, VRank computes the scores of $v_1$, $v_2$ and $v_3$ for Alice according to Equation (1):

$$R(v_1) = \alpha \sum_{i=1}^{S_{s_1}} (E(v_1, s_i) \text{Rank}(s_i)) + \beta \cdot F(v_1, P) \sum_{i=1}^{S_{s_1}} \text{Rank}(s_i)$$
$$= \alpha \cdot (0.15 + 0.24) + \beta \cdot 1 \cdot (0.15 + 0.24)$$
$$= 0.39 \cdot (\alpha + \beta)$$

(2)

Similarly, we also get the scores (i.e. threat levels) of $v_2$ and $v_3$: $R(v_2) = \alpha \cdot (0.24 + 0.35) + \beta \cdot 2 \cdot (0.24 + 0.35) = 0.59 \cdot \alpha + 1.18 \cdot \beta$, and $R(v_3) = \alpha \cdot (0.33 + 1.01) + \beta \cdot 3 \cdot (0.33 + 1.01) = 1.34 \cdot \alpha + 4.02 \cdot \beta$. If $\alpha$ and $\beta$ are both set to 0.5, Alice will receive a ranking list on vulnerabilities: $R(v_3) = 2.68$, $R(v_2) = 0.885$ and $R(v_1) = 0.39$.

**E. Another Possible Scoring Functions**

As mentioned above, Equation (1) considers the effects of exploitability and threat level separately before merging them together into a single score. Another possible choice is to take a holistic approach that considers the combined effect of both
in the first place. Specifically, if we interpret exploitability as directly related to the probability that a vulnerability will be exploited, then exploitability multiplied with its threat level would seem to naturally reflect the overall severity of a single vulnerability to a service. Based on this intuition, we present the scoring function as follows:

\[
R(v) = \sum_{i=1}^{[S_i]} E(v, s_i) \cdot \text{Rank}(s_i) \cdot F(v, P)
\]  

(3)

Admittedly, one can hardly argue only based on the definitions of scoring functions that one is always a better choice than the other. After all, the design of scoring functions relies on both reasonable security intuitions and one’s empirical experiences. Whether they are useful for real applications ultimately has to be evaluated through empirical study. In Section IV, we present three case studies aiming to show the effectiveness of our materialization of VRank.

F. Discussion

An obvious question is if existing vulnerabilities scoring systems can be used to rank vulnerabilities in SOA? Most of existing vulnerabilities scoring systems such as CVSS and CVE are very general, and they actually do not consider dependency of services which is the most important factor for SOA. They are actually more suit for scoring vulnerabilities in traditional (or general) software systems. Though CVSS recently introduces several mechanisms considering context-aware factors, the ones are not related to score vulnerabilities in SOA, and those approaches are not applicable in practice (at least do not work in SOA-based environment). Therefore, it is reasonable to think that current vulnerabilities scoring systems cannot address the problem of scoring vulnerabilities in SOA very well. Our VRank system can be looked as a specific approach to score vulnerabilities in SOA. It considers particular factors in SOA such as context-based information; thus, we can say our approach is more suitable for vulnerabilities scoring in SOA.

Another concern might be from the fact that some services (or softwares) might have been compromised by malicious attackers before ranking vulnerabilities, how to deal with this situation? For this practical issue, we present here a scheme which introduces TrustRank [13] instead of PageRank to calculate the importance and trust of each service, thereby discovering the services compromised by attackers; on the other side, TrustRank has the same capability as PageRank in terms of computing the importance of services. Therefore, introducing TrustRank to score services will not influence scoring vulnerabilities.

IV. Evaluation

This section mainly aims to evaluate the performance of VRank. We built a prototype system of VRank; then, through three case studies, we evaluate VRank’s effectiveness based on the contexts of real business processes and simulated vulnerability information. In addition, we compare the two proposed equations (i.e., Equation (1) and (3)) with CVSS, thus demonstrating the effectiveness of VRank. The steps of each case study are listed as follows:

1) We input concrete experimental data including both security requirements and the context of a business process into the prototype of VRank, thus obtaining a ranking list of vulnerabilities for the given security requirement. Notice that, in order to comprehensively evaluate our approach, we generate two ranking lists through Equation (1) and Equation (3) respectively; thus, we actually obtain two ranking lists by VRank.

2) We generate another ranking list by CVSS, and compare CVSS’s result with the previous two ranking lists provided by VRank’s two equations. The reason of using CVSS to compare with VRank will be mentioned later.

3) We compare and evaluate which ranking list is the best, by running our simulator which is designed to evaluate ranking lists of vulnerabilities. (The working principle of that simulator is given in Section IV-B.) Then, we analyze and discuss our results.

The rest of the this section firstly shows some discussions about CVSS, and then describes how to compare the different ranking lists produced by CVSS and our two approaches. Finally, we demonstrate the effectiveness of VRank based on three case studies.

A. Discussion on CVSS

With respect to CVSS, we need to show two important discussions: 1) why we choose CVSS to compare with VRank? and 2) how to treat basic and environmental metric of CVSS? The following will be answering the above questions.

Why compare with CVSS? There have been several vulnerability scoring systems in practice. Due to the fact that CVSS has been emerging as the standard in the community, we choose CVSS to compare with our approach. Furthermore, existing study in [12] also pointed that CVSS is currently a representative vulnerability scoring system.

How to deal with basic and environmental metric? In our following evaluations, we simulate CVSS with basic rating and the security requirement component of environmental rating mechanism. Actually, environmental metric includes three different components: Collateral Damage Potential (CDP), Target Distribution (TD) and Security Requirements (SR). However, because CDP and TD are not related to the target problem which we are focusing on, we do not need to consider those two parts when simulating CVSS. Moreover, according to the description in [4], many normal users are not going to use environmental metric and some other advanced rating schemes (e.g., temporal metric), because basic metric of CVSS should
be enough for almost all the users’ needs. Please see [4] for more details.

B. How to Compare Different Ranking Lists?

After getting three ranking lists (from CVSS, Equation (1) and (3)), we compare the three lists by running our simulator. We define performance metric as the average time to finish a task for a given business process. Here, we use tasks to denote sub-processes in a business process. For example, in Figure 4, a task is comprised of VCL access service, VCL management service and application service 3; and, a given business process contains all the components in Figure 4.

We assume the runtime of each service in the given business process is different; thus, when running a certain task in that business, we can obtain the task’s runtime by adding up all the runtime of services in that task. For example, in Figure 4, if we set runtime of VCL access service to 0.5, VCL management service to 0.5 and application service 3 to 0.4, then the task’s runtime should be $0.5 + 0.5 + 0.4 = 1.4$. Moreover, if a certain vulnerability is compromised, executing the services which has the compromised vulnerability will cost more runtime. Namely, we can think the compromised vulnerability delays the runtime of the task having that vulnerability. In our simulation, delay time is set to 0.2. For example, in Figure 4, if VCL access service has a compromised vulnerability, then the runtime of a task that is comprised of VCL access service, VCL management service and application service 3 should be $0.3 + 0.5 + 0.4 + 0.2 = 1.4$.

After describing how to compute performance metric, we present detailed steps of running our simulator:

1) We generate a simulation environment based on the given business process (including services, dependencies and vulnerabilities).

2) We compromise each vulnerability in the given business process respectively, and compute the performance metric with respect to each compromised vulnerability. Specifically, for each compromised vulnerability, we randomly execute different tasks 1000 times in the given business, and then compute average runtime of 1000 executions as the performance metric with respect to the compromised vulnerability.

After receiving performance metrics for different compromised vulnerabilities, we can evaluate which ranking list is better than others. It is obvious that the performance metric should be highest (i.e., delay time is the highest), if the most critical vulnerability is compromised. Say, if $v_3$ is the most critical vulnerability, performance metric of the simulator should be the highest. Therefore, after obtaining ranking list on all the performance metrics (the order is from high to low), we compare the matching degree between two ranking lists, i.e., ranking lists of vulnerabilities and ranking list of performance metrics (performance metrics are ranked from high to low). The more matching two ranking lists are, the more accurate the ranking list of vulnerabilities is. Next, we will present three case studies to evaluate our approach.

C. Case Study 1: Evaluation based on VCL Dataset

Data description. In this case study, we use a real business process from Virtual Computing Laboratory (VCL) to evaluate VRank. The information of business process is shown in Figure 4. The exploitability and threat levels (for given security requirement) of vulnerabilities are presented in Table I. Notice that we only extract business process from VCL, but not all the vulnerabilities. Due to privacy problem, we generated simulated vulnerabilities. Similarity, case study 2 and 3 also utilize real business processes, and vulnerabilities are simulated.

Scoring and ranking vulnerabilities. Based on two equations of VRank, i.e., Equation (1) and (3), we obtain two ranking lists with respect to vulnerabilities for the given security requirement; at the same time, we also generate another ranking list by using CVSS. Table II shows the three ranking lists. With respect to the values of $\alpha$ and $\beta$, we set $\alpha = 1.0$ and $\beta = 0.5$. Actually, we might also set other values to them (e.g., $\alpha = 0.5$ and $\beta = 0.5$), and obtain the same results; thus, we conclude that the values of $\alpha$ and $\beta$ in fact do not influence the results significantly.

<table>
<thead>
<tr>
<th>Exploitability</th>
<th>Threat Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2007-1747</td>
<td>0.801</td>
</tr>
<tr>
<td>CVE-2008-3648</td>
<td>0.45</td>
</tr>
<tr>
<td>CVE-2006-6077</td>
<td>0.3045</td>
</tr>
<tr>
<td>CVE-2008-5410</td>
<td>0.702</td>
</tr>
<tr>
<td>CVE-2008-0231</td>
<td>0.34</td>
</tr>
<tr>
<td>CVE-2008-0600</td>
<td>0.54</td>
</tr>
</tbody>
</table>

Table I: Case study 1: Exploitability & threat levels of vulnerabilities. We assume that the exploitability of the same vulnerability in different services is the same. Threat level is a value in range $[0, 5]$ under the given security requirement.
Evaluating ranking lists and discussion. To understand the accuracy of three ranking lists, our simulator is executed to evaluate effectiveness of three approaches. Figure 7a shows the results generated by our simulator which is designed to compare different ranking lists. As shown in Figure 7a, we can easily discover the performance metric is the highest when vulnerability CVE-2008-5410 is compromised. According to the performance metrics from high to low, we rank the vulnerabilities based on the performance metrics output by our simulator as: CVE-2008-5410, CVE-2008-0600, CVE-2007-1747, CVE-2006-6077, CVE-2008-3648, and CVE-2008-0231. Actually, this order correctly matches the ranking lists produced by VRank; on the other hand, due to failing to consider business context and threat level for the given security requirement, CVSS’s ranking list is inaccurate. Therefore, we can say experiment indicates VRank can work better than CVSS in the case study 1.

D. Case Study 2: Evaluation based on Business Process of EC2

Data description. In this case study, we make use of a real business process of EC2 [14]. Figure 5 shows concrete dependency relationships of services and vulnerabilities located on the business process. The exploitability and threat levels (for given security requirement) of the vulnerabilities are presented in Table III.

Scoring and ranking vulnerabilities. Similar with case study 1, we generate three ranking lists based on two equations of VRank, and the algorithm of CVSS. Table IV shows the three ranking lists.

Evaluating ranking lists and discussion. As shown in Figure 7b, after running our simulator, the result of our simulator shows the correct order of vulnerabilities should be: V-2009-3508, V-2007-6410, V-2009-4447, V-2006-6077 and V-2008-3631. On the other side, as shown in Table IV, we draw two conclusions: 1) the ranking list from Equation (1) is the most accurate, because it matches the results of our simulator the most; and 2) although the result of Equation (3) is accurate in the first case study, it fails to give a correct ranking list in this case study; meanwhile, the result from CVSS is also incorrect. Actually, this is because of the algorithms behind two approaches (i.e., Equation (3) of VRank and CVSS). Specifically, CVSS fails to consider dependency between services and Equation (3) has weak points at the aspect of algorithm. Therefore, we support Equation (1) of VRank can work better than other two approaches.

E. Case Study 3: Evaluation on IBM Disaster Assistance Claim Services

Data description. In this case study, we use data and information from IBM Disaster Assistance Claim Services. As shown in Figure 6, we understand concrete dependency relationships and vulnerabilities located on various services. The
Exploitability and threat levels (for given security requirement) of the vulnerabilities are presented in Table V.

**Scoring and ranking vulnerabilities.** Similar with previous two case studies, we generate three ranking lists. Table VI shows the three ranking lists.

**Evaluating ranking lists and discussion.** After running our simulator, as shown in Figure 7c, the result of our simulator presents the correct order should be: V-2009-2208, V-2009-2675, V-2010-9337, V-2009-9010, V-2010-4307 and V-2009-1031. We have three conclusions: 1) the ranking list from Equation (1) can rank vulnerabilities in this case study the most accurate, because it matches the results of our simulator the most; 2) the result of Equation (3) is also accurate but not the same accurate as Equation (1). In fact, we can say Equation (3) works better than CVSS in this case study at least; and 3) CVSS’s algorithm fails to obtain good result in this case study, because CVSS’s mechanism cannot get good result in the case of considering context of business process. Therefore, we support Equation (1) of VRank should be the most accurate, and can work better than CVSS in SOA.

V. RELATED WORK

In order to help security administrators, many vulnerability scoring systems have been proposed. Almost all of them mainly focused on how to score vulnerabilities based on two factors: impact and exploitability. It is obvious that they are not suit to be used in SOA, because they fail to consider contexts of concrete business process which is a key factor in SOA. Specifically, USCERT generates a quantitative severity score ranging from 0 to 180, computed directly from answers to a range of qualitative questions [15]. In addition, Microsoft’s Security Bulletin records vulnerability severity (with a qualitative approach) as do Secunia’s reports. Recently, a new severity metric, Common Vulnerability Scoring System (CVSS) [4], was generated by some security experts and researchers. CVSS defines several independent metrics; however, according to the study in [12], CVSS is only the “basic metric” which is typically used in third-party vulnerability databases. Compared with our approach, CVSS fails to provide any mechanism which explicitly takes into account contexts of businesses processes. In contrast, to overcome this particular challenge in SOA, VRank applies PageRank-like algorithm which is able to compute the importance of service in SOA. In fact, we further explore the semantic (i.e., concrete security requirements) of PageRank results which are used to calculate the importance of services in SOA.

In fact, there have been some previous efforts which were proposed to analyze and improve CVSS, e.g., [16] and [17]. Specifically, the work in [17] also considers context-based approach to score vulnerabilities; however, the main difference is the one uses context-based information to estimate administrator’s improvement on the approach of ranking vulnerabilities. In contrast, our approach uses context information to improve the accuracy of scoring vulnerabilities in SOA. Moreover, the effort proposed in [12] aims to score and predict vulnerabilities based SVMs algorithms; nevertheless, the approach also cannot be used to score vulnerabilities for composed services, since it did not consider some specific factors such as business contexts.

Currently, many reports on vulnerability disclosures have been published via multiple sources such as Bugtraq [18], Microsoft [19], CVE [5] and OSVDB [8]. Especially, CVE explicitly takes into account vulnerabilities’ different threat levels for any given requirements; unfortunately, CVE failed to provide reasonable mechanism for scoring vulnerabilities in SOA. Moreover, a number of studies have also examined the probability that vulnerabilities are able to be patched, e.g., [20] and [21].

VI. CONCLUSION

In this paper, we design a novel framework, VRank, to score and rank vulnerabilities for composed services in SOA. Different from existing efforts, for a given vulnerability, VRank

<table>
<thead>
<tr>
<th>Equation (1)</th>
<th>Equation (3)</th>
<th>CVSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>V-2009-2208</td>
<td>V-2009-2208</td>
<td>V-2010-4307</td>
</tr>
<tr>
<td>V-2009-2675</td>
<td>V-2009-2675</td>
<td>V-2009-1031</td>
</tr>
<tr>
<td>V-2010-9337</td>
<td>V-2010-9337</td>
<td>V-2009-2208</td>
</tr>
<tr>
<td>V-2009-9010</td>
<td>V-2010-9010</td>
<td>V-2009-9010</td>
</tr>
<tr>
<td>V-2010-4307</td>
<td>V-2009-1031</td>
<td>V-2010-4307</td>
</tr>
<tr>
<td>V-2009-1031</td>
<td>V-2009-1031</td>
<td>V-2010-9337</td>
</tr>
</tbody>
</table>

Fig. 6: A concrete business process from IBM disaster assistance claim services. Red rhombuses denote vulnerabilities. Others are different services in this business.
not only considers its intrinsic properties (e.g., exploitability), but further takes into account the contexts of those services having this vulnerability, e.g., which roles they play in the composed service and how critical it is to the security objective of the service. Through experiments, we evaluate VRank’s effectiveness, and compare VRank with CVSS, demonstrating VRank could work better than existing efforts in terms of scoring vulnerabilities in SOA.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their helpful suggestions. Jianchun Jiang is partially supported by the National Science and Technology Major Project under Grant No. 2010ZX01036-001-002 & 2010ZX01037-001-002, the Knowledge Innovation Key Directional Program of CAS under Grant No.KGCX2-YW-125 & KGCX2-YW-174and the National Natural Science Foundation of China under Grant No. 61170072. Ting Yu is partially supported by the NSF under Grant No.KGCX2-YW-125 & KGCX2-YW-174and the Knowledge Innovation Key Directional Program of CAS under Grant No. 2010ZX01036-001-002 & 2010ZX01037-001-002, the National Science and Technology Major Project under Grant No. 2010ZX01036-001-002 & 2010ZX01037-001-002, the Knowledge Innovation Key Directional Program of CAS under Grant No. 2010ZX01036-001-002 & 2010ZX01037-001-002, the National Science and Technology Major Project under Grant No.KGCX2-YW-125 & KGCX2-YW-174and the National Natural Science Foundation of China under Grant No. 61170072. Ting Yu is partially supported by the NSF under Grant No.KGCX2-YW-125 & KGCX2-YW-174.

REFERENCES