Highlighting Vulnerabilities via Context-Aware Framework

Eric Chang

Yale University

Abstract. It is known to be full of challenges to score vulnerabilities based on various security requirements in cloud services. Although there have been several systems for scoring vulnerabilities (e.g., CVSS), most of them are unable to be leveraged to score vulnerabilities in cloud services, because they fail to consider some important factors located in cloud such as business context (i.e., dependency relationships between services) and threat levels of the same vulnerability on various security requirements. This paper aims to propose a novel framework to qualify and rank the vulnerabilities based on their threat degrees in cloud service. Through inputting or constructing service dependency graph, our framework is able to generate the importance degree of each service and the ranking list of all the vulnerabilities in cloud service. Moreover, our framework can be adopted not only into various cloud infrastructures, but also different categories of algorithms according to concrete requirements. To evaluate our framework, we adopt AssetRank algorithm into the framework, and present the whole design of our work. Comprehensive experiments prove the effectiveness of our framework on qualifying and ranking vulnerabilities in cloud service.

1 Introduction

Background. Cloud computing has become increasingly popular as the next infrastructure for hosting large-scale data and being adopted application softwares [23], [24], [26], [18], [28]. Moreover, cloud computing also provides a new business model where software services and core computing are outsourced on demand to third-party infrastructure such as Amazon's Elastic Compute Cloud (EC2) [2], Microsoft's Azure Service Plantform [11], and Rackspace's Mosso [13]. Specifically, cloud infrastructure is able to provide personalized business (e.g., a concrete transaction in E-Business) to each client; meanwhile, a business is composed of many services such as searching goods and payment in E-Business. On the other hand, this new business model also introduces a range of risks [15]. Because there are many dependency relationships among different services, adversary is able to overwhelm a whole business through attacking the vulnerabilities of some *important services* contained in that business. Here, we use important services to denote the services which are located at the "critical paths" in dependency relationships among services, and call important services' vulnerabilities as principal vulnerabilities. Obviously, the threat aiming to compromise principal vulnerabilities will badly influence the availability and reliability of cloud service[1]. For

[1]We mainly use cloud service to denote a general concept on the services which are provided by different cloud infrastructures and companies.

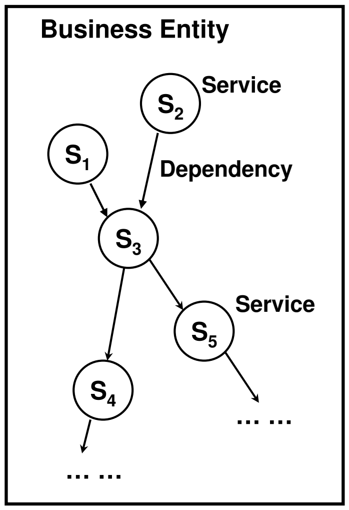

Fig. 1: A simple example presents the dependency relationships of services in a business entity, where: S_1, S_2, S_3, S_4 and S_5 are services, and each arrow denotes the dependency relationship between two different services. The whole business entity is composed of all the arrows and services.

example, the business entity shown in Fig. 1 is executed by user Alice, and adversary Bob can easily compromise the whole business through exploiting and attacking the vulnerabilities of service S_3 instead of attacking the vulnerabilities of other services (e.g., S_2, S_5). Therefore, computing and evaluating the threat degrees of vulnerabilities are known to be very important and challenging for current cloud services [7].

Previous efforts. In order to address the above problem, some analytical technologies have been proposed. Specifically, CVSS [1] provides an general approach to score IT vulnerabilities. SANS [16] releases TOP 20 list of IT vulnerabilities and many organizations have used that list to patch the most dangerous vulnerabilities. However, previous threat analytical approaches and systems mainly aim to design general technologies. Namely, they do not pay much attention to the concrete application scenarios such as specific services in cloud infrastructure. Thus, they will fail to be adopted or cannot provide the effective and reasonable estimations of vulnerabilities. In addition, some of previous work fail to consider the qualification of vulnerabilities in services. Although there are indeed several mechanisms aiming to rank vulnerabilities or detect threats were proposed (e.g., CVSS [1], and OWASP Top Ten [12]), these efforts cannot clearly present services' prior relationships, thus making security administrators (i.e., cloud providers) cannot understand the security situations of whole businesses in cloud very well. Therefore, it is still a "pendent question": *In order to enable cloud service to be more reliable, how to estimate and rank threat degrees of vulnerabilities in cloud service?*

Our approach. In order to overcome the above challenges, this paper presents a novel framework which can be adopted into current cloud infrastructures. Our framework aims to address the problem of qualifying and ranking threat degrees of vulnerabilities in cloud service. Specifically, cloud providers (e.g., security administrators of cloud companies) can input the dependency relationships among services of various businesses into our framework, thus obtaining the ranking list of threat degrees of vulnerabilities existing in the whole infrastructure. Receiving the ranking list of threat degrees will enable cloud providers to know how to deal with the vulnerabilities and apply defenses; moreover, the list is also very useful and helpful to normal users who aim to understand whether there are threats in cloud or not. Notice that our framework can be adopted with various algorithms used to compute the importance degrees, and the concrete situation should be based on different requirements of availability and reliability in cloud service. Namely, our approach can qualify, estimate and rank the threat degrees of vulnerabilities which exists in different application scenarios of cloud service. The operation of our framework contains three key steps:

1. Cloud providers construct and input service dependency graph for all the services of some businesses. Sometimes, constructing service dependency graph is completed automatically by some softwares and applications.
2. Our framework computes and ranks threat degrees of all the vulnerabilities of those services, thus generating the ranking list based on threat degrees and obtaining the importance degrees of services.
3. Service providers provide effective solutions to the vulnerabilities which are pointed to own relatively high threat degrees.

In addition, our framework is able to map each vulnerability to specific services affected directly by that vulnerability, and then generates a set of services influenced by the vulnerabilities of the services. Actually, the service dependency graph should be provided by other components in cloud infrastructure automatically, because the dependency relationships among services should have been constructed when the configuration of any business is completed. Providing solutions should be based on the concrete requirement of the availability and reliability of a concrete business, since the two parts (i.e., availability and reliability) are trade-off in real-world adoption. To the best of our knowledge, our framework is the first effort to automatically calculate the importance degrees of services and rank vulnerabilities based on their threat degrees.

The organization of this paper. The rest of this paper is organized as follows. Section 2 will give the problem statement, system model and attack model in order to make our target clear. We will present the details of our framework in Section 3. The design of our prototype system and our evaluations on the framework will be shown in Section 4 and Section 5, respectively. Section 6 aims to present and discuss related work, and conclusions will be drawn in Section 7.

2 Problem Statement and Model

In order to clear the description of our framework, this section mainly presents the problem statement of our work (in Section 2.1), system model (in Section 2.2) and attack model (in Section 2.3).

2.1 Problem Statement

Our design is a general framework which can be adopted into any cloud infrastructure, where: 1) there are many businesses, and each of them is composed of some services which contain or do not contain vulnerabilities, 2) there are three groups of identities in the system — normal users, cloud provider and attackers, and 3) normal users aim to execute their desired businesses, and we can also call them as clients of cloud service. We assume that normal users and cloud providers are completely trusted. Namely, attackers are the only adversary in the whole cloud service. As shown in Fig. 2, *services dependency graph* should be input into our framework. The framework computes the threat degree of each vulnerability according to those dependency relationships (i.e., services dependency graph), and outputs a *ranking list of vulnerabilities* to cloud providers or normal users. Responding to cloud providers or clients should depend on the concrete requirements. Notice that we use service dependency graph to denote the directed relationships among various services shown in Fig. 1, and we call a list which ranks various vulnerabilities based on threat degrees of those vulnerabilities as ranking list of vulnerabilities' threat degrees.

In particular, the design of our framework aims to answer the following three questions: 1) which vulnerability is the most important (or the most dangerous)? 2) how to obtain threat degrees of services' vulnerabilities in cloud service? and 3) how to rank these vulnerabilities coming from different services? Notice that our framework does not pay much attention to the concrete algorithm which is leveraged to compute the threat degrees of vulnerabilities. The goal of our framework is to provide an approach to qualify and rank the vulnerabilities of different services and report the concrete situation on security to cloud providers, thus enabling normal users to avoid attacks which are launched by exploiting those vulnerabilities. Moreover, how to generate service dependency graph should be provided by other components of cloud service, e.g., businesses generator. That should not be our target.

2.2 System Model

In order to illustrate our description clearly, this section mainly presents system model. For a typical cloud service system, there are a large number of users (or clients) $U = \{u_1, u_2, ..., u_n\}$ connecting to cloud and using different businesses. Moreover, they can also communicate or interact with each other.

There are many businesses in cloud service. We use $B = \{B_1, B_2, ..., B_n\}$ to denote the set of businesses. Each business, actually, is composed of many services. We model the whole business (i.e., only a certain business) as a directed random graph $B_i = (BS_i, A_i)$, where BS_i is a set of service tuples, and the set contains all the services in the ith business B_i. Notice that we define S as the set of all the services. Obviously, we can have $BS_i \subseteq S$. A_i is defined to be the set of all the dependency (directed relationships) among services in the business B_i. For each BS_i, we have $BS_i = \{bs_1, bs_2, ..., bs_n\}$ and service tuple is denoted to be $bs_j = \langle serv_j, Vul_j \rangle$, where $serv_j$ is a tuple which contains related information of the jth service in BS_i, and Vul_j is a set which includes different vulnerabilities of the jth service in BS_i.

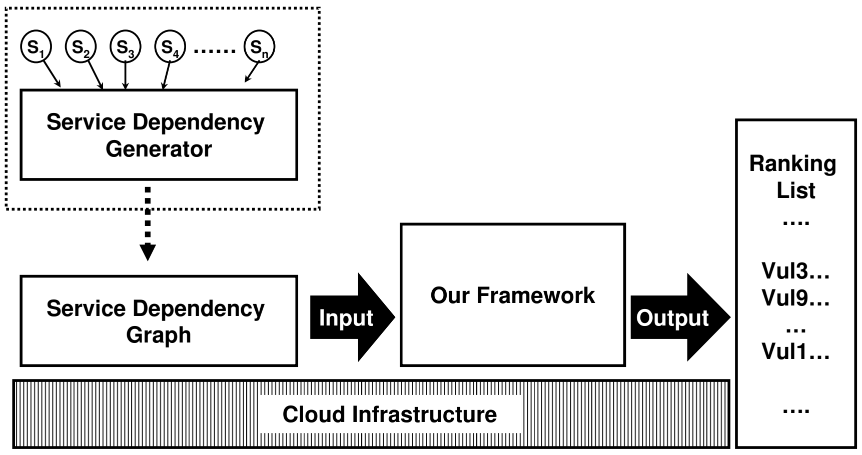

Fig. 2: The overview of our framework. Services S_1 through S_n are used to be input into the generator of service dependency graph. The whole service dependency graph is input into our framework and then, based on service dependency graph, the concrete algorithm in the framework is able to generate a ranking list on the threat degrees of vulnerabilities.

Specifically, $serv_j = \langle ID, Owners, ..., Content\rangle$, where ID is this service's ID number, $Owners$ is the owners and groups of this service, and $Content$ is the description of the service's functions. $Vul_j = \langle v_1, v_2, ..., v_n\rangle$, where v_k denotes the kth vulnerability in the whole set of vulnerabilities (i.e., Vul_j). A_i, which is defined as the set of all the dependency among different services, is the set of directed edges from service bs_m to another service bs_n, where $m \neq n$. For a typical user u_i, we define the user's behavior $ub_i \in UB$ as the set $ub_i = \{e_{i,1}, e_{i,2}, ..., e_{i,n}\}$, and define each $e_{i,x} \in ub_i$ as the tuple $e_{i,x} = \langle u_i, B_x\rangle$. The tuple means a user u_i perform an execution to the business B_x which is located in cloud service.

2.3 Attack Model

In a typical cloud service, all the subjects (i.e., participants) can be grouped into three categories: normal users, cloud service providers, and attackers (i.e., adversary). Generally, a normal user can also be called client, and he or she should be a honest user. Attackers can be some computers controlled by adversary aiming to compromise some components (e.g., services and businesses) in cloud service. We assume that cloud providers are completely trusted, and adversary cannot compromise the hosts of cloud providers. Adversary can only launch attacks by using their computers or through controlling botnets networks [14].

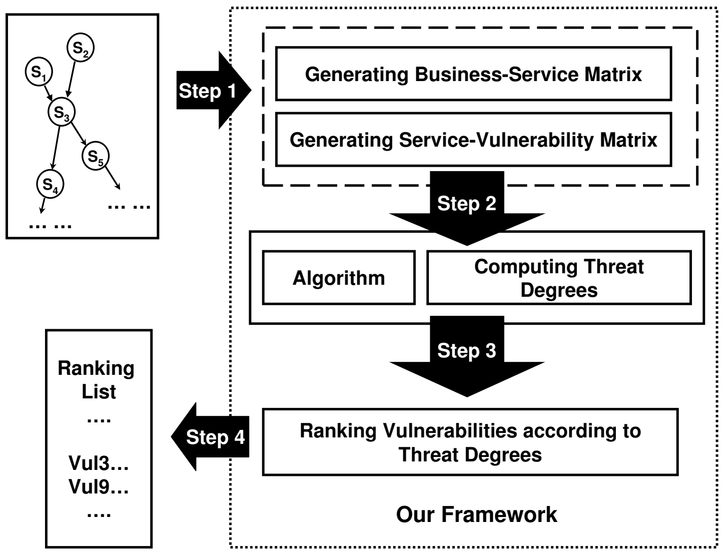

Fig. 3: The concrete adoptions in our Framework. Actually, as the input component of our framework, service dependency graph is firstly mapped to two matrixes. Through using two matrixes, the algorithm is able to qualify the threat degree for each vulnerability existing in the services which are from service dependency graph. In final step, a ranking list based on threat degrees of vulnerabilities is produced by our framework.

Normal users. We assume there are N normal users in cloud service. Their purposes are only to perform businesses in cloud, and they can be compromised by attackers.

Malicious users. We assume that malicious users are able to know what vulnerabilities are located in any given service, and what services are contained by a concrete business. Notice that we do not make any assumption of the numbers of normal users and attackers, since the two numbers are not related to our design and evaluations.

3 The Design of Our Framework

This section mainly presents the design of our framework. In our experiments, we adopt AssetRank algorithm into our framework. Namely, AssetRank algorithm is leveraged as the algorithm of threat degree in our framework. Therefore, the description of our framework is mainly based on AssetRank. Notice that AssetRank is only an instance of the actual utilization of our framework, and other algorithm computing importance degrees can also be adopted into the framework. Fig. 3 presents the details of components contained in our framework.

3.1 Background: AssetRank [17]

AssetRank algorithm is a generalization of the PageRank algorithm mainly used to rank web pages in web graphs. The study in [17] shows that AssetRank is able to address the unique semantics of dependency attack graphs and incorporates vulnerability data from public databases to compute metrics for the graph vertices which reveal their importance in attacks against the system. The algorithm treats the whole networking as a graph, and it is mainly used to be adopted to rank the importance degrees of a vertex in a dependency graph. Each numeric value computed by AssetRank is a direct indicator of how important the attack asset represented by a vertex is to a potential attacker. In fact, this category of rank-based metric will be valuable to users of attack graphs in better understanding the security risks, in fusing publicly available attack asset attribute data, in determining appropriate mitigation measures, and as input to further attack graph analysis tools.

Actually, there are four contributions in AssetRank [17]. The first contribution is that AssetRank can correctly treat the AND and OR vertices in a dependency attack graph based on their logical meanings which cannot be done by PageRank algorithm [5]. The second contribution is the generalization of AssetRank which allows AssetRank to accurately model the various likelihoods of an adversary's capability to obtain privileges through means not captured in the graph. The third contribution is that using publicly available vulnerability information (e.g. CVSS [1]) enables the importance degree of security problems to be computed on vulnerability attributes such as attack complexity. The fourth contribution is that the generalized ranking algorithm allows network defenders to obtain personalized AssetRank to reflect the importance of attack assets about the protection of specific critical network assets. Based on the above advantages, we adopt AssetRank into our framework to compute the threat degrees of vulnerabilities in cloud service. On the concrete details of AssetRank algorithm, please see [17].

3.2 Calculation of Threat Degrees of Vulnerabilities

In our framework, because any given business is constructed by a group of services, any service (say, s_i) should be a component of one business entity. According to system model defined in Section 2.2, we define BS_i as the set of services in $B_i \in B$ (i.e., B_i denotes the ith business in the set of businesses). We use $\{bs_j\}_{j=1}^{|BS_i|}$ to denote each service in BS_i. In addition, it is easy to know $BS_i \subseteq S$.

Preparation work. From now on, our description will always be with respect to the calculation of the threat degree of a particular vulnerability v_f. In fact, for this vulnerability (i.e., v_f), we need to know which businesses are influenced by it. However, we cannot obtain that information directly in the real system. In our work, we firstly need to know which services are influenced by the vulnerability v_f, and define $SV_f = \{sv_{f1}, sv_{f2}, ..., sv_{fn}\}$ to be the set of services containing vulnerability v_f. We know that each business is composed of a group of services. If the vulnerability v_f exists in a certain service and that service is located in some businesses, all the businesses will be affected by v_f. Thus, through traversing all the businesses, we use

$BV_f = \{bv_{f1}, bv_{f2}, ..., bv_{fn}\}$ to denote the set of businesses including the vulnerability v_f. Because vulnerability v_f is located in multi-service and these services should be depended on multi-business. Therefore, vulnerability v_f will influence one or more businesses and the threat degree of v_f should be based on the importance degrees of multi-services.

Based on the above description, we need to obtain two vectors in order to achieve our final result (i.e., the threat degree of vulnerability v_f). The first vector is used to record which services are located in BS_i, and the other one aims to record the services which own the vulnerabilities in v_f. Notice that because our computation needs to traverse each business, we will obtain different BS_is. The first vector's target is to record the concrete services in each different BS_i. For each business in BV_f (say, B_k), we use $f(\cdot)$ to denote the first vector, and use $h(\cdot)$ to denote the second one. The concrete equation generating the first vector is shown as Equation (1).

$$f(BS_k, s_i) = \begin{cases} 1 & s_i \in BS_k \\ 0 & s_i \notin BS_k \end{cases} \tag{1}$$

In order to know the threat degree of a certain vulnerability, we need to know what services contain that vulnerability. As defined previously, for any service $s_i \in SV_f$, we can obtain the second vector by using Equation (2):

$$h(SV_f, s_i) = \begin{cases} 1 & s_i \in SV_f \\ 0 & s_i \notin SV_f \end{cases} \tag{2}$$

After obtaining two vectors, we can compute the importance degree of Business k through Equation 3. In our work, we can also use impact of Business k to denote the importance degree of Business k.

$$Impact(B_k) = \frac{\sum_{i=1}^{|S|} f(BS_k, s_i) \cdot h(SV_f, s_i) \cdot score(s_i)}{\sum_{i=1}^{|S|} f(BS_k, s_i) \cdot score(s_i)} \tag{3}$$

where:

– $score(s_i)$: the importance degree of service s_i computed by AssetRank algorithm.

Through traversing all the businesses, we use $BV_i = \{bv_1, bv_2, ..., bv_n\}$ to denote the set of business containing the vulnerability v_i. By using Equation 3, we need to compute all the $Impact(bv_i)$s (i.e., $Impact(bv_1)$, $Impact(bv_2)$, ..., $Impact(bv_n)$), and then compute the threat degree of each vulnerability v_i in Business k, through using Equation 4.

$$Threat(v_i) = \sum_{j=1}^{|BV_i|} Impact(B_j) \tag{4}$$

Actually, it is easy to be found that the final results $Threat(v_f)$ must be computed based on all the above equations. In order to illustrate our approach clearly, we will present an example to illustrate the process of computing a concrete $Threat(v_i)$.

3.3 An Example

There is a service set $S = \{s_1, s_2, s_3, s_4, s_5\}$ in company A and these services construct the following business sequences: $B_1 = \{s_1, s_2, s_3\}$, $B_2 = \{s_2, s_5\}$ and $B_3 = \{s_1, s_3, s_4\}$. The percentages of B_1, B_2 and B_3 are: $B_1 = 20\%$, $B_2 = 30\%$ and $B_3 = 50\%$. We assume that v_1 is located in service s_1 and s_2, and v_2 is located in service s_2. Businesses B_1, B_2 and B_3 will be influenced by vulnerability v_1, because B_1 and B_3 contain s_1 and B_2 contains s_2. Thus, we can use Equation (5) to compute the above example.

$$\begin{cases} Threat(v_1) = Impact(B_1) + Impact(B_2) + Impact(B_3) \\ Impact(B_1) = \frac{score(s_1)+score(s_2)}{score(s_1)+score(s_2)+score(s_3)} \\ Impact(B_2) = \frac{(s_2)}{score(s_2)+score(s_5)} \\ Impact(B_3) = \frac{score(s_1)}{score(s_1)+score(s_3)+score(s_4)} \end{cases} \tag{5}$$

$\Rightarrow$

$$\begin{aligned} Threat(v_1) =& \frac{score(s_1) + score(s_2)}{score(s_1) + score(s_2) + score(s_3)} \\ &+ \frac{score(s_2)}{score(s_2) + score(s_5)} \\ &+ \frac{score(s_1)}{score(s_1) + score(s_3) + score(s_4)} \end{aligned} \tag{6}$$

Similarly, we also know the threat degree of v_2:

$$\begin{aligned} Threat(v_2) =& \frac{score(s_1)}{score(s_1) + score(s_2) + score(s_3)} \\ &+ \frac{score(s_1)}{score(s_1) + score(s_3) + score(s_6)} \end{aligned} \tag{7}$$

We obtain the importance degree of each service in $S = \{s_1, s_2, s_3, s_4, s_5\}$ by using AssetRank. The concrete relationships and importance degrees are shown in Fig. 4, and we have known the importance degree of each service (i.e., s_1, s_2, s_3, s_4 and s_5): $score(s_1) = 0.0387$, $score(s_2) = 0.0423$, $score(s_3) = 0.1703$, $score(s_4) = 0.6517$ and $score(s_5) = 0.097$. We can calculate the threat degree for each vulnerability as follows: $Threat(v_1) = 0.322 + 0.304 + 0.045 = 0.671$ and $Threat(v_2) = 0.154 +$

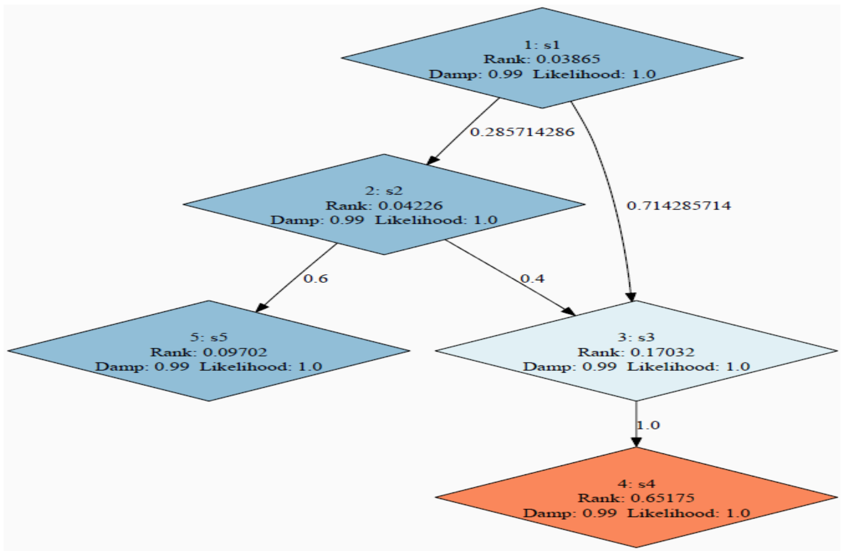
1: s1
Rank: 0.03865
Damp: 0.99 Likelihood: 1.0
0.285714286
0.714285714
2: s2
Rank: 0.04226
Damp: 0.99 Likelihood: 1.0
0.6
0.4
5: s5
Rank: 0.09702
Damp: 0.99 Likelihood: 1.0
3: s3
Rank: 0.17032
Damp: 0.99 Likelihood: 1.0
1.0
4: s4
Rank: 0.65175
Damp: 0.99 Likelihood: 1.0

Fig. 4: Our Example: The Dependency Relationships between Services.

$0.045 = 0.199$. Based on the above results, we know how each vulnerability influence businesses and services. We reason the threat degree coming from vulnerability v_1 is larger than vulnerability v_2. Therefore, we complete qualifying the threat degrees for vulnerabilities.

4 Implementation of Our Framework

This section mainly describes the design and implementation of our framework. Because our framework has not been adopted in real cloud infrastructure yet, we give the principal architecture and functions of each component in our prototype system.

Framework architecture. As shown in Fig. 5, the architecture of our framework is composed of four principal components: 1) the acquisition of service dependency data, 2) the computation of importance degrees of services, 3) the computation of threat degrees of vulnerabilities, and 4) ranking manager. The description of each component is listed as follows:

– **The acquisition of service dependency data.** This component aims to generate service dependency data (e.g., information on dependency graph). In fact, the data can also be given from other components or modules in the system, because service dependency graph is generated by other functional components automatically in some cloud service systems.

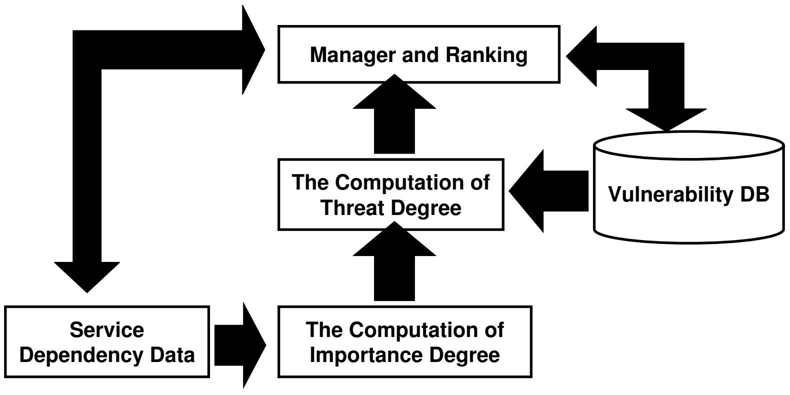
Manager and Ranking
The Computation of Threat Degree
Vulnerability DB
Service Dependency Data
The Computation of Importance Degree

Fig. 5: Components Structure of Our Implementation.

- **The computation of importance degrees for services.** This component calculates the importance degree for each service after it receives the service dependency data. In our current framework, we use AssetRank to realize that function.
- **The computation of threat degrees of vulnerabilities.** This component contains three main functions: 1) to locate which services have special vulnerabilities, 2) to locate which business will be influenced by those special vulnerabilities, and 3) be used to calculate the threat degree of each vulnerability.
- **Ranking manager.** This component is used to manage the above results and rank vulnerabilities based on their threat degrees.

Main functions of implementation. In order to automatically calculate the threat degrees of vulnerabilities in an given business, we firstly need to know which services are influenced by a certain vulnerability. In our implementation, we design an function for mapping between vulnerability and service, and we name that function as VulnerabilityMapService. Besides the above work, we still should know which business is affected by existing vulnerabilities of services. The implementation in this paper designs another function for mapping existing vulnerabilities of services to some businesses. That function is called ServiceMapToBuiness. After knowing those existing vulnerability services and businesses, the implementation will be given the threat degree for each vulnerability.

5 Evaluation

To evaluate our framework, we use real data from Virtual Computing Laboratory (VCL) to generate the ranking list on the threat degrees of vulnerabilities. AssetRank algorithm

Table 1: Distribution of Business and VCL Services

	S_1	S_2	S_3	S_{71}	S_{20}	S_{35}	S_{59}	S_9	S_7	S_{68}
Business1	1	1	1	0	0	0	0	0	0	0
Business2	1	1	0	1	0	0	0	0	0	0
Business3	1	1	0	0	1	0	0	0	0	0
Business4	1	1	0	0	0	1	0	0	0	0
Business5	1	1	0	0	0	0	1	0	0	0
Business6	1	1	0	0	0	0	0	1	0	0
Business7	1	1	0	0	0	0	0	0	1	0
Business8	1	1	0	0	0	0	0	0	0	1

is adopted as concrete mechanism of our framework. Then, we build a simulation experimental environment to verify whether the ranking list on the threat degree of vulnerabilities is accurate or not. Thus, we can say the evaluation mainly includes two steps: 1) the generation of ranking list with respect to the threat degree of vulnerabilities, and 2) evaluate that ranking list by using simulation.

5.1 VCL Environment

In order to evaluate our design, we introduce the real data from VCL into our experiment. Notice that due to the data's confidentiality, we have to adjust some of the data. There are four principal components contained in VCL: 1) end-user access interface, 2) VCL manager, 3) an image repository, and 4) computational, storage and networking hardware. Because we aim to evaluate vulnerabilities' security problems in VCL, we describe the VCL as a group of services from end-user perspective:

1. **VCL access service.** It is responsible for the service to provide an end-user access interface which is usually implemented by Linux, Apache and PHP.
2. **VCL management service.** The management service is responsible for all VCL resources such as blades, virtual machines and lab machines; meanwhile, it contains a scheduler, multi-site coordination, performance monitoring, and virtual network management.
3. **VCL application service.** These services are implemented by all application images which can be scheduled by VCL Management.

The above services in VCL work together to accomplish business coming from the end-user. The logical dependency graph of those services is shown in the Fig. 6.

5.2 Ranking VCL's Vulnerabilities

In this section, we use real data to rank threat degrees of vulnerabilities in VCL. In order to guarantee privacy, we have processed and modified parts of data. We obtained reservation information on Feb. 18th VCL Application Services. The number of VCL application services for user request from reservation information can be extracted. Our main work is based on the following three steps:

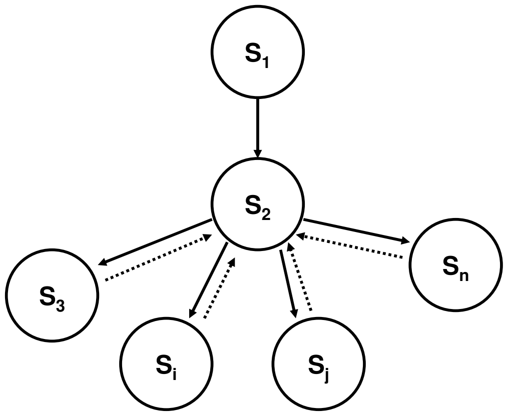

Fig. 6: Services dependency graph in VCL. s_1 denotes VCL access service. s_2 is used to denote VCL management services, and $s_3 - s_n$ is defined as VCL application services. A end-user accesses VCL through VCL access service s_1 to select from a menu, and a combination of applications, operating systems and VCL application services ($s_3 - s_n$) that are needed. VCL management service s_2 maps that requested image(s) onto available (possibly heterogeneous) hardware resources.

Table 2: Distribution of Vulnerabilities and Services

	S_1	S_2	S_3	S_{71}	S_{20}	S_{35}	S_{59}	S_9	S_7	S_{68}
CVE-2007-1747	0	0	1	0	0	0	0	0	0	0
CVE-2008-3648	0	0	0	1	0	0	0	0	0	0
CVE-2006-6077	0	0	0	0	1	0	0	0	0	0
CVE-2008-5410	0	1	0	0	0	1	0	0	0	0
CVE-2008-0231	0	0	0	0	0	0	1	0	0	0
CVE-2008-0600	0	1	0	0	0	0	0	0	0	0

Table 3: Ranking List of Vulnerabilities

Vulnerability Name	**Threat Degree**
CVE-2008-5410	6.323253
CVE-2007-1747	0.052587
CVE-2008-0600	0.052587
CVE-2008-3648	0.046974
CVE-2006-6077	0.035546
CVE-2008-0231	0.018309

1. Based on the collection of the data, we compute these services dependency strength and generate the dependency data as the input of our framework, and according to the functions of our implementation, we can get the importance degree of each service in VCL. We established one matrix for businesses and services, as shown in Table 1.
2. We know the vulnerability will exist in different VCL services. Based on Section 4, we built another matrix with vulnerabilities and services from CVE, as shown in Table 2. In order to protect privacy of VCL, vulnerabilities in Table 2 are all assumed.
3. We use the framework to compute threat degree for each vulnerability and rank them. We make BSMatrix and VSMatrix as input of our framework and then automatically rank threat from these vulnerabilities, as shown in Table 3.

From the above tables, we know that the biggest threat degree comes from the vulnerability named CVE-2008-5410. We analyze that CVE-2008-5410 vulnerability exists in VCL service S_2 and all businesses depend on the service S_2. If we assume that one vulnerability just only affect one VCL service, the result about ranking threat from these vulnerabilities have the same treat degree as service important value, as shown in Fig. 7. Actually, we can easily get that our approach to rank threat vulnerabilities of VCL is strongly related to business and these ranking result will be helpful for security administration of images in VCL.

5.3 Evaluating Ranking List via Simulation

Goal of simulation. This section aims to answer a question: *whether we can believe the ranking list from our framework?* Actually, any algorithm or mechanism is able to provide a ranking list on vulnerabilities in a business, and it is also entirely possible that cloud provider and clients cast doubt to the ranking list produced by our framework. For example, Alice and Bob are able to provide two different ranking lists based on the threat degrees of vulnerabilities in the same cloud service. *How can we judge which is correct?* Therefore, how to prove whether our ranking list is correct or not is a principal goal for this section.

Why choose simulation? We choose to use simulation to answer the above question based on the following reasons: 1) Simulation experiments enable us to study the performance of large-scale systems. Because most cloud companies tend to protect users'

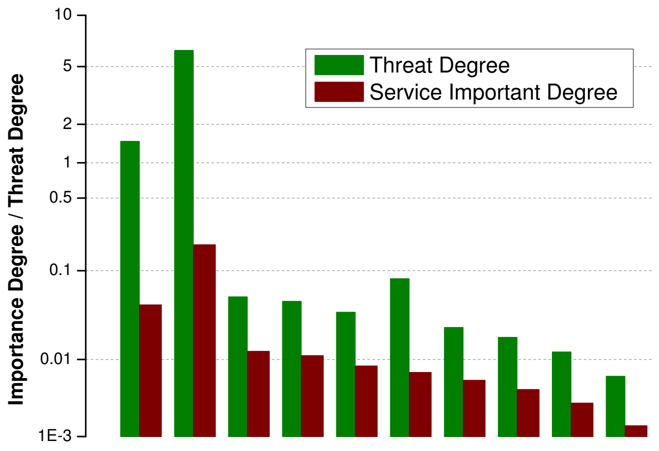

Fig. 7: Importance Degree V.S. Threat Degree

private data, there are only a limited number of publicly available real-world datasets; 2) Simulation experiments enable us to generate our wanted attacking forms which cannot be obtained from the real-world traces; 3) We generate experimental environments totally based on the measurement results of real-world and practical distributions. Therefore, the authenticity and persuasion of simulation experiments are able to be insured completely. *To sum up, experiments based simulation can fulfill the goals of our evaluations.*

Details of our simulation. Our simulation-based experiment is mainly composed of three steps:

1. **Step I.** We build simulation environment (i.e., infrastructure) based on the description on EC2 in [2], and adopt real data (e.g., service entity and vulnerabilities shown in Table 1) into our environment.
2. **Step II.** We attack each vulnerability respectively, and respectively calculate the total performance[2] of all the businesses after compromising every vulnerability. Namely, when we attack the first vulnerability, we compute the total performance of all the businesses (shown in Table 1) as the result of attacking the first vulnerability, and then calculate the whole performance when launching attack to the second vulnerability as the second result, and so on. Thus, we are able to obtain many different results of performances under the situation that various vulnerabilities are attacked.

[2]We use the sum of the performances from all the businesses to denote the total performance.

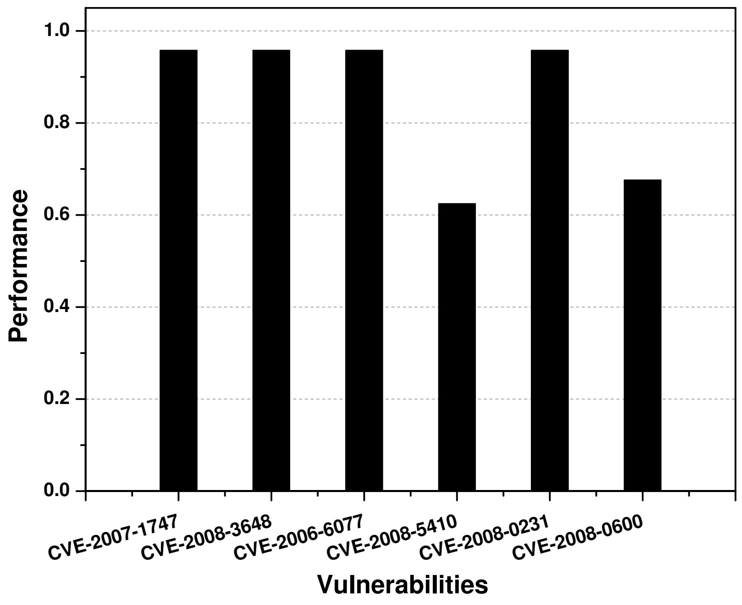

Fig. 8: The Performance of Simulation.

3. **Step III.** We compare those results with our ranking list. It will prove our ranking list is correct if the low to high order of performance results matches our ranking list — the high to low order of threat degrees of vulnerabilities.

Notice that the performance is only a metric in our simulation. It has no any concrete semantic, and, actually, we do not need to give the performance any concrete semantic, because our simulation only focuses the order of total performances but not the meaning of those performances.

Discussion. As shown in Fig. 8, we can see the total performance is the lowest under the adversary attacks vulnerability CVE-2008-5410. According to the low to high order, we can rank the vulnerabilities based on the total performances in our simulation as: CVE-2008-5410, CVE-2007-1747, CVE-2008-0600, CVE-2008-3648, CVE-2006-6077 and CVE-2008-0231. Actually, this order correctly matches the ranking list produced by our framework. Therefore, we can say our ranking list is proved to be correct by our simulation experiment.

6 Related Work

Our work mainly focuses on ranking vulnerabilities in cloud service based on their threat degrees. There are some previous efforts aiming to address this problem. The CVSS [1] is a general approach which proposes standardized mechanism to rate IT vulnerabilities; however, CVSS fails to provide any mechanism being able to calculate businesses' impacts in the application scenario of cloud service. On the other hand, in order to overcome this problem, our approach applies AssetRank algorithm which is able to achieve this goal in cloud service. In fact, we further explore the semantic of AssetRank results which are used to calculate the importance degrees of services in cloud service.

Moreover, many efforts (e.g., [4], [3], [8], [6], [30], [27], [22], [29], [10], [19], [20], [25], [21], [9]) aiming at the dependency relationships discovery of services have been developed. Specifically, Sujoy Basu et al. [4] used automatical identification of dependency traces of messages to discover dependency relationships of Web services and is able to detect the dynamics of those services' relationships. Paul Barham et al. [3] proposed Constellation which applies the timings of packet transmission and reception to infer the complicated relationships between hosts and services. Christian Ensel presented an approach to automatically generate service dependency models which is implemented by agents [8]. Xu Chen et al. [6] implemented Orion system and used packet headers and timing information in network traffic to discover dependencies. However, these previous efforts only concentrated on how to discovery service dependency relationships. In contrast, our approach differs from them and we study how to leverage services' dependency relationships to rank all the vulnerabilities according to their threat degrees in cloud service.

7 Conclusion

In this paper, we design a novel framework to qualify and rank the threat degrees of vulnerabilities. Our contributions mainly present an approach which is used to calculate both threat degrees of vulnerabilities and importance degrees of services. As an instance, we adopt AssetRank algorithm into the framework. After present the whole design of our framework, we use simulation experiments to prove the effectiveness of the framework with respect to qualifying and ranking vulnerabilities in cloud service.

References

1. A Complete Guide to the Common Vulnerability Scoring System. URL:. `http://www.first.org/cvss/cvss-guide.html`.
2. Amazon Elastic Compute Cloud (EC2). URL:. `http://aws.amazon.com/ec2/`.
3. Paul Barham, Richard Black, Moises Goldszmidt, Rebecca Isaacs, John MacCormick, Richard Mortier, and Aleksandr Simma. constellation automated discovery of service and host dependencies in networked systems. Technical Report MSR-TR-2008-67, Microsoft Research, Apr 2008.
4. Sujoy Basu, Fabio Casati, and Florian Daniel. Toward web service dependency discovery for soa management. In *IEEE SCC (2)*, pages 422–429, 2008.
5. Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.
6. Xu Chen, Ming Zhang, Zhuoqing Morley Mao, and Paramvir Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *OSDI*, pages 117–130, 2008.

7. Yanpei Chen, Vern Paxson, and Randy H. Katz. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley, Jan 2010.
8. Christian Ensel. A scalable approach to automated service dependency modeling in heterogeneous environments. In *EDOC*, pages 128–139, 2001.
9. Jianchun Jiang, Liping Ding, Ennan Zhai, and Ting Yu. Vrank: A context-aware approach to vulnerability scoring and ranking in SOA. In *Sixth International Conference on Software Security and Reliability, SERE 2012, Gaithersburg, Maryland, USA, 20-22 June 2012*, pages 61–70, 2012.
10. Bo Liu, Ennan Zhai, Huiping Sun, Yelu Chen, and Zhong Chen. Filtering spam in social tagging system with dynamic behavior analysis. In *2009 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2009, 20-22 July 2009, Athens, Greece*, pages 95–100, 2009.
11. Microsoft Azure Services Plantform. URL:. `http://www.microsoft.com/azure/default.mspx/`.
12. OWASP Top Ten. URL:. `http://www.owasp.org`.
13. Rackspace Mosso. URL:. `http://www.mosso.com/`.
14. Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. My Botnet is Bigger than Yours (Maybe, Better than Yours): Why size estimates remain challenging. In *USENIX HotBots*, 2007.
15. Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM Conference on Computer and Communications Security*, pages 199–212, 2009.
16. SANS Top-20 Security Risks. URL:. `http://www.sans.org/top20`.
17. Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *ESORICS*, pages 18–34, 2008.
18. Cong Sun, Ennan Zhai, Zhong Chen, and Jianfeng Ma. A multi-compositional enforcement on information flow security. In *Information and Communications Security - 13th International Conference, ICICS 2011, Beijing, China, November 23-26, 2011. Proceedings*, pages 345–359, 2011.
19. Yonggang Wang, Ennan Zhai, Cui Cao, Yongqiang Xie, Zhaojun Wang, Jian-bin Hu, and Zhong Chen. Dspam: Defending against spam in tagging systems via users' reliability. In *16th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2010, Shanghai, China, December 8-10, 2010*, pages 139–146, 2010.
20. Yonggang Wang, Ennan Zhai, Jian-bin Hu, and Zhong Chen. Claper: Recommend classical papers to beginners. In *Seventh International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010, 10-12 August 2010, Yantai, Shandong, China*, pages 2777–2781, 2010.
21. Yonggang Wang, Ennan Zhai, Eng Keong Lua, Jian-bin Hu, and Zhong Chen. isac: Intimacy based access control for social network sites. In *9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, UIC/ATC 2012, Fukuoka, Japan, September 4-7, 2012*, pages 517–524, 2012.
22. Ennan Zhai, Ruichuan Chen, Zhuhua Cai, Long Zhang, Eng Keong Lua, Huiping Sun, Sihan Qing, Liyong Tang, and Zhong Chen. Sorcery: Could we make P2P content sharing systems robust to deceivers? In *Proceedings P2P 2009, Ninth International Conference on Peer-to-Peer Computing, 9-11 September 2009, Seattle, Washington, USA*, pages 11–20, 2009.
23. Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. An untold story of redundant clouds: making your service deployment truly reliable. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems, HotDep 2013, Farmington, Pennsylvania, USA, November 3, 2013*, pages 3:1–3:6, 2013.

24. Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. Heading off correlated failures through independence-as-a-service. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, pages 317–334, 2014.
25. Ennan Zhai, Liping Ding, and Sihan Qing. Towards a reliable spam-proof tagging system. In *Fifth International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, 27-29 June, 2011, Jeju Island, Korea*, pages 174–181, 2011.
26. Ennan Zhai, Liang Gu, and Yumei Hai. A risk-evaluation assisted system for service selection. In *2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, June 27 - July 2, 2015*, pages 671–678, 2015.
27. Ennan Zhai, Zhenhua Li, Zhenyu Li, Fan Wu, and Guihai Chen. Resisting tag spam by leveraging implicit user behaviors. *PVLDB*, 10(3):241–252, 2016.
28. Ennan Zhai, Qingni Shen, Yonggang Wang, Tao Yang, Liping Ding, and Sihan Qing. Secguard: Secure and practical integrity protection model for operating systems. In *Web Technologies and Applications - 13th Asia-Pacific Web Conference, APWeb 2011, Beijing, China, April 18-20, 2011. Proceedings*, pages 370–375, 2011.
29. Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. Spamclean: Towards spam-free tagging systems. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, Vancouver, BC, Canada, August 29-31, 2009*, pages 429–435, 2009.
30. Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. Sorcery: Overcoming deceptive votes in P2P content sharing systems. *Peer-to-Peer Networking and Applications*, 4(2):178–191, 2011.