A Risk-Evaluation Assisted System for Service Selection

Ennan Zhai[†] Liang Gu[†,‡,⋆] Yumei Hai[‡]
[†]*Department of Computer Science, Yale University, US*
[‡]*Department of Computer Science, Jinan University, China*
ennan.zhai@yale.edu, os.liang.gu@gmail.com, haiyumei@gmail.com

***Abstract*—With the rapid adoption of Service Oriented Architecture (SOA), increasingly more application-level services are developed through composing service components offered by different service providers. While such application development mode offers advantages in terms of cost-effectiveness and flexibility, application developers cannot understand or deal with risks potentially resulting from vulnerabilities within composed services due to non-transparency of the service providers. Furthermore, some of the vulnerabilities in practice are deeply hidden in dependency structures underlying composed services, thus making even the service providers fail to know the vulnerabilities.**

This paper proposes a risk-evaluation assisted service selection system, called RiskEvaluation-as-a-Service(or REaaS), which aims to assist application developers to understand vulnerability risks hidden within alternative services when the developers at first attempt to adopt their applications. In particular, for a given application developer's service selection requirement, REaaS produces a ranking list based upon vulnerability risks of alternative services to serve as a guideline regarding which service has the lowest potential risks (e.g., bugs) for this application deployment. REaaS achieves this goal through the following three steps: 1) generating a package dependency graph for each alternative service, 2) assigning threat-degrees to packages in each dependency graph, and 3) analyzing each dependency graph and evaluating service-risk of each service. We have built a REaaS prototype and used real case study to demonstrate the practicality of REaaS.

I. Introduction

Service Oriented Architecture (SOA) concepts and techniques have already transformed the Internet into an important service delivery platform rather than mere a global communication provider. It has been explicit that an increasing trend for organizations and enterprises to develop sophisticated application-level services by composing decentralized services offered by different providers (either public or private), which introduces significant advantages in terms of cost-effectiveness, extensibility and flexibility.

Nevertheless, recent efforts [18, 2] indicated that such a shift of paradigm potentially introduces new security risks. In particular, different services used or composed by application-level services in SOA are often developed by various providers, which are built upon diverse third-party packages. Thus, the types of vulnerabilities faced by application providers are much more diverse than the ones in traditional applications. There have been public vulnerability databases containing: 1) classification of vulnerabilities, 2) description of the nature of their threats, and 3) some severity scores based on metrics proposed by security experts. However, it is still extremely challenging for the application developers to detect and identify fault-cause due to non-transparency of those service providers.

In order to tackle this problem, we propose to use risk-evaluation based service selection approach to help the application developers understand potential risks resulting from vulnerabilities within each of alternative services. This effort intuitively enables the application developers to avoid selecting the services holding "dangerous" vulnerabilities. In fact, service selection technique in SOA is not a new topic yet. While there have been many service selection techniques, these approaches mainly focus on choosing suitable services through measuring qualities of services (i.e., QoS) rather than taking risks caused by vulnerabilities into account. Furthermore, most of the current service selection efforts fail to offer systematic solutions.

Following the above insight, this paper designs and builds a practical risk-evaluation assisted service selection system, called RiskEvaluation-as-a-Service(or REaaS), for application developers. To estimate potential risks, REaaS first automatically extracts package dependencies within alternative services, and assigns threat-degree to each package within the services through leveraging existing vulnerability scoring systems such as CVE [3] and CVSS [4]. Then, REaaS quantifies service-risk for each of the alternative services and finally offers a ranking list based on these service-risks to serve as a guideline regarding which service holds the lowest vulnerability risks for application developers.

Besides the above design, building a practical REaaS still needs to address a few challenges in real world. With the temptation of getting more users, service providers may not honestly report the configuration of their services. Meanwhile these providers might be concerned with the leakage of their sensitive information such as internal package dependencies. We employ TPM-based attestation to present evidences of these operations which enables both service providers and REaaS to attest the authenticity of their executions. With this design, service providers are able to verify if their internal information has been leaked and REaaS also can check whether the service providers honestly perform its protocol.

Our REaaS prototype has a few limitations and would require further development to support more sophisticated service selections in SOA. Nevertheless, we believe this first-step effort suggests a practical path towards complementing current SOA service selection techniques. This paper makes the following contributions. 1) To the best of our knowledge,

[⋆]Corresponding Author

we are the first to study the problem of service selection based on evaluations of risks resulting from vulnerabilities. 2) We present a novel design, REaaS, which provides the first practical solution to this problem. 3) We build a REaaS prototype and demonstrate the practicality of our system through evaluating this prototype with realistic case study.

II. System Model

This section describes system model including: target scenario (Section II-A) and trust assumptions (Section II-B).

A. Target Scenario

Our target scenario consists of three different types of entities: clients, service providers and service-selection system.

Clients. The clients are run by application developers who plan to deploy their applications upon one or more services offered by multiple individual service providers. A client, i.e., application developer, in practice might be a large organization with vast resources and global interests such as Netflix [7] or an individual with limited resources and regional interests such as Zynga [10]. In our scenario, each of clients aims to select one or more services which have the lowest risks from many alternative service providers having similar functions to deploy her application.

Service providers. The service providers host and offer software related supports for storage, communication or computational purposes. With respect to a similar service function, there are normally many service providers. They need to register on a certain service-selection system (defined later), thus forming a set of alternative services to potential clients. The goal of service providers is to sell their services to clients without leaking their internal information such as package dependencies within their services.

Service-selection system. The service-selection system fills the gap between clients and service providers. A client who attempts to adopt her application needs to express her expectations in terms of submitting a requirement to the service-selection system who in turn makes a service-selection suggestion obtained based upon its service-selection scheme. The service-selection system aims to serve as a guide who recommends clients the most suitable services according to their requirements.

B. Assumptions

Clients (or application developers) are honest but curious. Each of them will faithfully participate in a service-selection process, but may try to exploit additional information that can be learned in doing so. The clients do not operate fake identities, which means there is no collusion (i.e., information sharing) among clients.

Service providers are potentially malicious in that they expect to benefit themselves by dishonestly registering their information on the service-selection system. For instance, a service provider may enable its service to obtain better service-selection score, if it does not register information which might lead to negative impact on its score calculations. Since the providers desire selling their services, the service providers have motivations to join the service-selection system in practice. However, the service providers do not allow the service-selection system to leak internal information within their services to the clients. Namely, only the service-selection system can keep the internal information of each service provider.

The service-selection system is potentially malicious in that it may try to leak service providers' internal information by analyzing services, but the service-selection system faithfully performs its own service-selection protocol. The service-selection system cannot create fake identities, and there is no collusion between clients and the service-selection system.

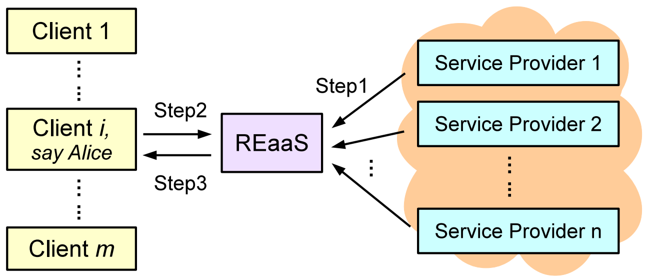


Figure 1: A typical REaaS service selection process, where an application developer, Alice, running Client i wants to select a service with the lowest vulnerability risks from n alternative service providers for her application.

III. Basic System Design

In order to provide a risk-evaluation assisted service selection, we make a systematic effort – designing and developing a practical system, RiskEvaluation-as-a-Service(or REaaS), which is applicable to the deployment model given in Section II.

This section presents REaaS's basic design under assumption that all the three entities (i.e., clients, REaaS and service providers) are honest but curious. In the next section, we will present how to use trusted platform module attestation based technique to enhance this basic design, thus enabling the REaaS to handle scenario under assumption that REaaS and service providers are potentially malicious (as mentioned in Section II-B).

A. System Working Process

We now use Figure 1 to illustrate how does REaaS assist a given client, say Alice, to select a service from many alternative service providers.

Step 1: Service provider registrations. First of all, service providers who want to sell or publish their services need to contact a REaaS and register on the REaaS. These registered service providers would show as alternative services to clients who are potential customers of the REaaS. In REaaS's service registration, each service provider needs to collect dependency information about packages underlying its service, and uses the collected information to construct a dependency graph. Namely, a dependency graph in principle represents an alternative service. Then, each service provider sends its dependency graph to the REaaS as registering information. Our REaaS offers package dependency collection and dependency graph generation tools to service providers. Details about service registrations are given in Section III-B.

Step 2: Requirement submission. After registration phase, a client, say Alice (Client i in Figure 1), who plans to deploy her application upon one or more services for storage, communication or computation purpose, contacts the REaaS. She conveys the REaaS her requirements which typically include: 1) which alternative services she may use, and 2) what security objective she is concerned with. The former item is a set of alternative service providers from which Alice wants to select one or more services. For the latter one, REaaS provides three optional security objectives: integrity, confidentiality and availability. Alice could make her choice according to her application type and domain. In practice, Alice may not have enough knowledge for the above requirements, so she is allowed to consult and negotiate with the REaaS.

Step 3: Service-risk evaluation. After receiving requirement from Alice, the REaaS performs service-risk evaluation to produce a *service-risk value* for each of alternative services. This value is used to quantify and account for potential risks resulting from vulnerabilities within a service. Until finish service-risk evaluations to all the alternative services, the REaaS generates a list by ordering these services based on their service-risk values. Then, the REaaS sends the generated ranking list back to Alice. With this list, Alice is able to understand potential vulnerability risk underlying each of alternative services, thus making the selection for her application under this guide. Detailed designs on REaaS are mentioned in Section III-D.

B. Service Provider Registrations (Step 1)

We now present design of each step in REaaS working process. In the initial phase, each of the service providers P_i needs to submit their registration information to the REaaS. Besides each service's name, function description and unique identity, service providers also need to submit REaaS registration information on package dependencies within their services. Thus, each provider, P_i, performs a collection operation to its own service S_i in order to extract such information. This information would be used by REaaS to make service-risk evaluations. Note that clients cannot see package dependency information within each of alternative

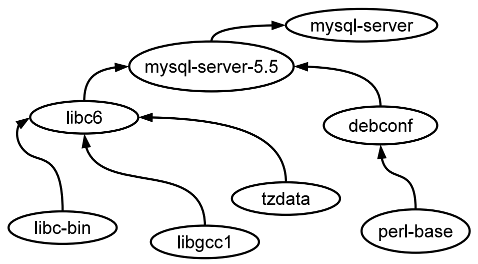


Figure 2: Dependency graph corresponding to MySQL package dependency example.

services, since the REaaS needs to preserve the privacy of internal information of each service provider.

Automatically extracting complete information about package dependencies within services is extremely challenging in practice. In REaaS design, we adapt our previous effort indaas-depends, the automatic package dependency collector of INDaaS [27], and offer this adapted collector to service providers. A provider P_i inputs her service S_i to the adapted indaas-depends. The indaas-depends searches through the apt-cache and the apt-rdepends [1] to find out all the package components and their dependencies within S_i. The indaas-depends presents a listing of each dependency a package has, and then iterates each of these packages and recursively lists their dependencies. As a result, the adapted indaas-depends emulates all the dependencies by organizing them in terms of a dependency graph G_i. Namely, a dependency graph in principle represents a target service.

We now use a simple but illustrative example to show a package dependency collection result and its corresponding dependency graph that are output by our adapted indaas-depends. This example assumes that we are trying to discover and collect package dependencies within MySQL database service, and the following is the package dependency collection result.

```
mysql-server
  Depends: mysql-server-5.5
mysql-server-5.5
  Depends: libc6 (>= 2.14)
  Depends: debconf (>= 0.5)
debconf
  Depends: perl-base (>= 5.6.1-4)
libc6
  Depends: libc-bin (= 2.15-0ubuntu10)
  Depends: libgcc1
  Depends: tzdata
```

The above result forms a dependency graph, as shown in Figure 2, where each node represents a package. This dependency graph will be used by service-risk evaluation.

C. Requirement Submission (Step 2)

After initial service registration phase, clients who want to select services for their applications are allowed to send requirements to the REaaS. As mentioned in Section III-A, a given client's requirement mainly includes two items. The first item is used to list a set of services which have been registered on the REaaS as alternative. The second one aims to explicitly describe security objective (or concern) on evaluating risks within the alternative services. For instance, if some client wants to select a storage service only for her application's exception event logging, which is rarely invoked, then the client may desire a storage service with the lowest integrity risks rather availability or confidentiality. In the REaaS design, we provide three options on security objectives: availability, integrity and confidentiality, since CVE + CVSS can provide the three types of risk estimations (shown in Section III-D).

D. Service-Risk Evaluations (Step 3)

When receiving a given client's service selection requirement, the REaaS performs service-risk evaluation to each of alternative services S_i, and produces a service-risk value quantifying potential risk with respect to S_i. Once the REaaS finishes service-risk evaluation to all the alternative services, it will generate a list ranking these services based on their service-risk values.

The REaaS produces a service-risk value for S_i through three main modules: 1) package threat-degree assignment, 2) risk-weighted dependency graph generation, and 3) service-risk quantification. The input of the REaaS is a dependency graph G_i representing structures underlying the target service S_i.

Package threat-degree assignment. With the package dependency graph G_i as input, the REaaS utilizes its first module, package threat-degree assignment, to assign a *threat-degree* to each package used by S_i. Since each node in G_i represents a package of S_i, the module iterates all the nodes in G_i. For each node (i.e., package), the module finds out all of its public vulnerabilities through querying *CVE + CVSS*. The Common Vulnerabilities and Exposures (CVE) system [3] provides a database on publicly known information-security vulnerabilities and exposures. Different from CVE, the Common Vulnerability Scoring System (CVSS) [4] offers an open framework for quantifying impacts of public vulnerabilities. Thus, CVE + CVSS store all the public vulnerabilities for a given package and their base-scores. The base-scores are generated based on some sophisticated equations formulated by CVSS security supervisors. In the REaaS, we use TD_i to denote threat-degree of a given package i. TD_i is the average of base-scores of all the vulnerabilities within the package i. Equation 1 presents the detailed computation of TD_i.

$$\begin{cases} TD_i = \sum_{j=1}^{n} BS_{j(i)}/n \\ BS_{j(i)} = (0.4 \cdot Exp_{j(i)} + 0.6 \cdot Imp_{j(i)}) \cdot 1.176 \\ Imp_{j(i)} = 10.41 \cdot ImpactLevel_{j(SecObj)} \end{cases} \tag{1}$$

Where, $BS_{j(i)}$ means base-score of the jth vulnerability of package i. The computation of base-score of a vulnerability is similar to the one in CVSS, which depends on two parameters: $Exp_{j(i)}$ and $Imp_{j(i)}$. The former one, i.e., $Exp_{j(i)}$, can be directly obtained by searching vulnerability j in CVSS. We only adapt the computation of $Imp_{j(i)}$, which is responsible for computing the vulnerability j's impact on client's specified security objective $SecObj$. We use $ImpactLevel_{j(SecObj)}$ to denote the impact level of vulnerability j on security objective $SecObj$. This impact level could be extracted from CVSS according to specific $SecObj$ which is given in client's service selection requirement. Note that numbers (e.g., 1.176, 10.41, 0.4 and 0.6) in Equation 1 are set by CVSS.

Risk-weighted dependency graph generation. After obtaining threat-degree of each package within S_i, the second module of the REaaS attaches these values to their corresponding node in G_i, thus forming a risk-wighted package dependency graph. We use *risk-weighted* G_i to denote this new dependency graph.

Service-risk quantification. The REaaS's third module, service-risk quantification, evaluates the risk of S_i by performing PageRank algorithm [21] on the risk-weighted G_i. Weight of each node in the PageRank computation is threat-degree of the package corresponding to the node. Intuitively, if many packages with high risks support a service, the service is likely to play a dangerous role in the alternative service set. Based upon PageRank algorithm, the service-risk quantification module is able to get a service-risk with respect to S_i under the specified security object. Of course, other dependency graph analysis algorithms such as HITS [17] can also be adopted to compute service-risks instead of using PageRank. Furthermore, if the dependency graph has richer semantic features, the importance computation function could be enhanced accordingly. Using which algorithm is an option for our system, and the decision may be made based upon different applications. In general, we believe that PageRank is one representative importance function that is suitable to our purpose and is expected to be useful to estimate service-risks of services.

E. Ranking List Generation

Running the above three steps can obtain the service-risk of one service. Once finishing computations of service-risks of all the alternative services, the REaaS generates a ranking list which orders the services according to their service-risk values. Then, the REaaS sends the ranking list back to the client. Since the client receives only an ordered list reflecting potential risks within alternative services, she obtains no proprietary information on the service providers' internal package information from this result. In addition, with this ranking list, the client can understand the possible risk of each service under her security concern; thus, she can pick out the most secure service.

IV. TPM-Based Attestation Enhanced REaaS

Last section presents REaaS basic design under assumption that all the entities are honest but curious. However, in practice, REaaS and service providers might be malicious, which have been defined in Section II-B. Namely, REaaS might leak service providers' registration information and service providers may dishonestly submit their registration information to REaaS. Thus, this section presents a full version of REaaS which is able to handle the scenario with assumption that REaaS and service providers could be potentially malicious.

In order to guarantee that REaaS does not leak the service providers' information to clients or any other unauthorized parties, REaaS employs Trusted Platform Module (TPM) based attestation to support the remote attestation on its execution. This enables each of service providers to get the corresponding execution records on REaaS's service-risk evaluation, and the providers can verify whether the risk evaluation process strictly follows the protocol and does nothing more. Similarly, each service provider also needs to deploy TPM locally in order to support REaaS's verification on whether the provider honestly registers its information such as collected package dependency information.

Before presenting how do we design TPM-based attestation enhanced REaaS, we first show a preliminary about general TPM technology and attestation in Section IV-A.

A. Preliminary: Trusted Platform Module Technology

A series of Trusted Computing specifications were introduced to provide the hardware-based trust bootstrapping on a platform. Among all of them, the Trusted Platform Module (TPM) [25, 24] is the fundamental building block, a secure co-processor widely deployed on desktops, laptops and servers. In order to guarantee its role as the root of trust, TPM is required to be implemented with tamper-resist techniques to protect it from physical attacks. To offer a strong identity, the TPM uses an Attestation Identity Key (AIK). Based the Storage Root Key (SRK), TPM can build up a hierarchy key system to provide storage protection and key management. All other typical security mechanisms can be built on top of this hierarchy key layer, such as authorization and message authentication. To track the hash values that constitute a fingerprint, the TPM provides special registers called Platform Configuration Registers (PCRs). Whenever a reboot occurs, the PCRs are reset and then can be used to record new hash values.

Remote attestation is one of the most important features presented by the Trusted Computing specifications. With an authenticated booting procedure, TPM can automatically record the configuration of a platform, from BIOS to the Operating System and finally up to the applications' states. In a typical TPM-based attestation scenario, a challenger, who wants to know whether some application operates honestly, sends a challenge request to platform running the application. The platform uses TPM to sign all the state records relative to the target application and then sends the signed state records back to the challenger. The challenger can verify the configuration of the platform/application by checking all the records, and the genuine of these records can be attested by checking the hash chain.

TPM-based attestation provides a reliable way for parties in different trust domains to check the platform configuration. As the progress of research on remote attestation, it is practical to extend the TPM-attestation to attest semantic properties of systems[11, 15]. In this paper, REaaS employs TPM-based attestation to provide evidences for service providers to verify the execution process of the REaaS. With the support of remote attestation, REaaS is able to prove whether it does preserve registration information for the service providers. Similarly, due to the attestation support, REaaS also can check whether service providers honestly submit their registration information such as package dependency information.

B. Remote Attestation for REaaS

Remote attestation on service providers' behaviors. In Step 1, after a service provider collects its own package dependencies, it already can submit the collected information to the REaaS as registration information. Nevertheless, as what we mention in trust assumption section (Section II-B), service providers might want to hide parts of their information to get better scores in the REaaS's service-selection process. In order to ensure each of service providers honestly submits its package dependency information, the REaaS employs TPM-based remote attestation protocol. This design needs each of service providers to run the package dependency collection operation on a TPM equipped machine. Under TPM-based attestation, all the states that each service provider performs package dependency collection would be recorded and signed, thus forming a specific hash-chain. Then, the REaaS could attest the honesty of service providers by checking the integrity of their hash-chains, since the hash-chains record all the states of the service providers' dependency collection operations.

Remote attestation on REaaS's behaviors. In practice, the REaaS is potentially malicious in that it may leak service providers' registration information such as internal package dependencies. To make sure that the REaaS will do as it is specified in the protocol and nothing more, TPM-based attestation is employed to attest the REaaS's execution. We now present how to perform a remote attestation to a service-risk evaluation execution of REaaS.

On the REaaS side, the system is required to be equipped with TPM. When the REaaS boots up, an authenticated boot process is employed and the states of all these loaded modules are recorded as R_{boot} by TPM PCRs before loading REaaS on the platform. Namely, R_{boot} contains the system information before running REaaS. When the REaaS prepares to run a risk evaluation with respect to some service, say S_i, the state of REaaS needs to be recorded with TPM, denoted as R_{REaaS}. Meanwhile, the registered package dependency information of the service S_i is also recorded with TPM, denoted as $R_{depinfo}$. When the evaluation is running, TPM enables the monitoring in the system to record all inputs and outputs of the REaaS. We use R_{input} and $R_{outputs}$ to denote these two types of records. After the service evaluation, the REaaS employs TPM PCR to record the generated S_i's service-risk (denoted as SR), and binds it with all the previous records by signed quote: $S_{evl} = Sig\{ R_{boot} || R_{REaaS} || R_{depinfo} || R_{input} || R_{outputs} || SR\}$. All the above records are organized as a hash-chain and are sent to the provider of S_i for verification. The service provider could examine all these records to confirm whether its internal information is leaked or whether any unspecified behaviors were done by the REaaS.

V. Evaluations

In this section, we evaluate REaaS system. We have developed a REaaS prototype which implemented all of our designs (described in Section III and Section IV). The prototype consists of $2,700$ lines of Python code. Using the prototype, we mainly aim to answer two questions: 1) whether REaaS is capable of assisting application developers to select the services with the lowest risks, and 2) how is the performance of REaaS? In order to answer the first question, we construct a realistic case study which helps a given application developers to select service with the lowest vulnerability risks. Then, we measure performance of REaaS by comparing running time of basic version REaaS (i.e., REaaS without TPM-based attestation) with full version REaaS (i.e., REaaS equipped with TPM-based attestation).

A. A Case Study

To explore the practicality of our system, i.e., ability on selecting service, we present a realistic case study using the REaaS prototype to assist an application developer to select a storage service.

1) Constructing Our Case Study: Alice, a rising entrepreneur in video on demand (VoD), intends to deploy and publish a video application service. Alice in practice may have limited infrastructure resources and lack of experience on building sophisticated storage systems for her application. Thus, she wants to adopt a storage service which were developed and offered by existing storage providers in order to support the data storage of her application. In real world, there are many alternative storage services, so Alice does not know which one has the best availability. In this case study, we assume Alice is the most concerned with the availability of alternative storage services. Luckily, she was recommended REaaS, a system that is able to assist her to pick out suitable service based on her requirement (availability in this case study). Alice therefore decides to use REaaS to select a storage service from four alternative storage service providers which have been registered on the REaaS.

Table I: Deeper Analysis Results.

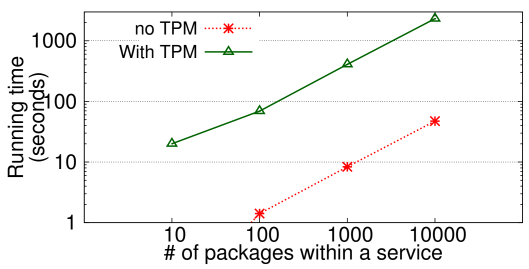
	MySQL	PostgreSQL	Riak	MongoDB
# of packages	588	736	103	108
Service-Risk	8	7	4	2

In our case study, we construct four storage service providers using four real and well-known storage systems: Riak [9], MongoDB [5], MySQL [6], and PostgreSQL [8], respectively. We use Service Provider1-4 to denote providers who offer the above storage services, respectively. Name, Service Provider1 offers Riak, Service Provider2 offers MongoDB, Service Provider3 offers MySQL, and Service Provider4 offers PostgreSQL. While realistic storage service providers might use other different storage systems (e.g., OracleDB and DB2), this configuration sufficiently exercises our REaaS prototype. Note that Alice in practice does not know what storage systems used by alternative service providers in practice.

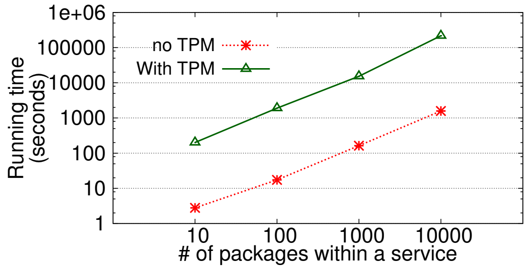
2) Evaluating Practicality of REaaS: We now evaluate the practicality of our REaaS prototype by following steps which are consist to the phases shown in Section III-A.

Step 1: Service provider registrations. In order to register the information on the REaaS, four storage service providers begin collecting package dependencies underlying their services using indaas-depends deployed in our REaaS prototype. With indaas-depends, four service providers obtain package dependency information and generate the corresponding dependency graphs. Note that this collection action is under TPM-based attestation. After the data collection, each of the service providers sends its registration information including identity, service name, functions and package dependency graph to the REaaS.

Step 2: Requirement submission. After that, Alice begins her process by installing a REaaS client on her machine. In order to select a storage service, Alice generates a requirement including: 1) alternative storage service providers, i.e., service provider1-4 in this case study and 2) security objective: availability, which is thought to be the most



(a) Running time about performing package dependency collection.



(b) Running time about performing a service-risk evaluation.

Figure 3: Running time measurement (using and without TPM)

important factor for selecting storage services. Then, Alice sends this requirement to the REaaS.

Step 3: Service-risk evaluation. After receiving Alice's requirement, the REaaS begins service-risk evaluation operations to four alternative storage services which have been registered on the REaaS. Under the assistance of CVE + CVSS, the REaaS first computes threat-degrees of all the packages underlying the four alternative services, thus generating a weighted package dependency graph for each of the services. Then, the REaaS gets service-risk with respect to each service by running the PageRank algorithm (with $1,000$ iterations) on weighted package dependency graph representing the service. Once finishing the computations of all the services' service-risks, the REaaS produces a list which orders the four alternative storage services according to their service-risks, and sends the ranking list to Alice. In our experiment, the ranking list generated by our REaaS prototype is:

```
Four Alternative Storage Services
---------------------------------
    1. Service Provider3
    2. Service Provider4
    3. Service Provider1
    4. Service Provider2
---------------------------------
```

The ranking list means MySQL (i.e., Service Provider3) has the highest service-risk and MongoDB (i.e., Service Provider2) has the best availability.

3) Result Analysis: In order to show deeper evaluational results, we now expose more information during the process of using REaaS in our case study. Table I presents service-risk of each of alternative storage systems. Note that these service-risks are not shown in the list Alice receives from the REaaS. The service-risks are obtained by performing PageRank on weighted package dependency graphs representing the four services. Using indaas-depends, each of alternative service providers can extract package dependencies underlying their storage systems. As shown in Table I, we observed the number of packages of PostgreSQL and MySQL are much higher than Riak and MongoDB. This is because both PostgreSQL and MySQL are relational database systems which need many package and library supports; on the contrary, Riak and MongoDB are NoSQL storage systems which normally need less packages. In addition, we observe that while MySQL has less packages than PostgreSQL, MySQL's service-risk is the highest.

B. Performance Evaluation

We now evaluate performance of REaaS by measuring our prototype's running time. Two of the most important operations are package dependency collection and service-risk evaluation. Thus, this section measures the running time of these two operations respectively.

We first measure running time that a service provider generates a local package dependency graph representing its service. Since this operation needs to be attested by TPM, we measure both cases, i.e., running time with and without TPM. The case without TPM could be looked as a microbenchmark of extracting package dependency information from a given service. In our measurement, we generated services holding N packages, and set $N = 10$, 100, $1,000$ and $10,000$ respectively. The evaluation results are shown in Figure 3a. Similarly, we then measure running time that REaaS performs a service-risk evaluation to a given service with and without TPM. We also vary the number of packages held by the target service, N, between 10 and $10,000$. The running time without TPM case could be looked as a microbenchmark of performing service-risk evaluation to a service. Figure 3b presents the experimental results.

For our experiments, we used one Dell XPS14 laptop with 2.8GHz 4-Core Intel Xeon CPU and 16GB memory. The PageRank computation in service-risk evaluations, we use $1,000$ iterations.

As shown in Figure 3a and Figure 3b, we observed that TPM attestation is a performance bottleneck in our system. It introduced a lot of computational overhead to REaaS working process. Nevertheless, given the two operations are executed off-line in practice, we believe that the overhead is acceptable to our system and clients.

VI. Related Work

To the best of our knowledge, REaaS is the first systematic effort which offers service selection through quantifying potential risks resulting from vulnerabilities.

Vulnerability-based analysis in SOA. The most representative vulnerability analysis approach in SOA is ATLIST [20]. ATLIST uses dependency graph model such as fault tree [22] to represent dependencies between various vulnerabilities in a given SOA business process. With this dependency graph, the administrator holding the SOA business process is able to analyze and find out the most important vulnerabilities. Different to our approach, ATLIST is a tool generating vulnerability-based dependency graph for the target SOA business process; in contrast, our approach aims to select "secure" services for application developers. In addition, ATLIST failed to provide practical analysis algorithm or a systematic effort. Similar to ATLIST, Jiang et al. proposed VRank [16] to rank vulnerabilities within any given SOA business process. Rather than recommending services to consumers, VRank offers fine-grained way to score vulnerabilities based on their service contexts. However, the purpose of VRank is different from REaaS, because VRank only focuses on vulnerability scoring.

Reputation-based service selection. There are a number of efforts using trust and reputation techniques to select services in SOA [12, 23, 14]. The web service selection approach proposed by Wang and Vassileva [26] generates reputation scores with respect to each alternative service based on feedback (e.g., votes) from communities or agencies. Similarly, Galizia et al. [13] proposed an approach to compute reputation values of services based on ontology mapping technique. Nevertheless, almost all the existing reputation-based service selection efforts assume a trusted third-party, and the proposed approaches have not been developed in practice yet.

Policy-based service selection. Policy-based service selection techniques in SOA allow to specify requirements by producing a QoS policy. Liu et al. [19] proposed a method which uses a trusted third-party to collect property information of all the services and select the most suitable services for consumers based on their policies which are expressed in forms of matrix. Regrading policy-based service selection efforts, there are some problems. First, it is difficult to formalize all the non-functional criteria in order to allow overall score estimation. Second, all the properties have to be presented as numbers which is not straightforward in practice. Third, it is challenging how to represent the values of properties. Like reputation-based techniques, there has not been a systematic effort in practice yet.

VII. Conclusion

In this paper, we design and develop a practical system, REaaS, to estimate risks of alternative services for application developers. Different from existing efforts which select services based on quantifying their QoS, our system aims at quantifying potential risks resulting from vulnerabilities within services. In order to design REaaS, we leverage the following methods: software package dependency collection, dependency graph analysis, vulnerability disclosure systems and TPM-based attestation. Furthermore, we built a first-step REaaS prototype system and use realistic case study to evaluate the prototype.

Acknowledgments

We thank anonymous reviewers for their insightful comments. This research was supported by the National Science Foundation of China under Grant NO. 61402198.

References

[1] apt-rdepends. http://packages.debian.org/sid/apt-rdepends.
[2] Common-mode software failures. http://www.lawfareblog.com/2014/04/heartbleed-as-metaphor/.
[3] Common Vulnerabilities and Exposures (CVE). http://cve.mitre.org.
[4] Common Vulnerability Scoring System (CVSS). http://nvd.nist.gov/cvss.cfm.
[5] MongoDB. http://www.mongodb.org/.
[6] MySQL. http://www.mysql.com/.
[7] Netflix. https://signup.netflix.com/.
[8] PostgreSQL. http://www.postgresql.org/.
[9] Riak 1.3.1. http://basho.com/riak/.
[10] Zynga. https://zynga.com/.
[11] Liqun Chen, Rainer Landfermann, Hans Löhr, Markus Rohe, Ahmad-Reza Sadeghi, and Christian Stüble. A protocol for property-based attestation. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing*, STC '06, pages 7–16, New York, NY, USA, 2006. ACM.
[12] Nelly A Delessy and Eduardo B Fernandez. A pattern-driven security process for SOA applications. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 416–421. IEEE, 2008.
[13] Stefania Galizia, Alessio Gugliotta, and John Domingue. A trust based methodology for web service selection. In *Semantic Computing, 2007. ICSC 2007. International Conference on*, pages 193–200. IEEE, 2007.
[14] Carlos Gutiérrez, Eduardo Fernandez-Medina, and Mario Piattini. Pwssec: process for web services security. In *Web Services, 2006. ICWS'06. International Conference on*, pages 213–222. IEEE, 2006.
[15] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation—a virtual machine directed approach to trusted computing. In *the Third virtual Machine Research and Technology Symposium (VM '04). USENIX.*, 2004.
[16] Jianchun Jiang, Liping Ding, Ennan Zhai, and Ting Yu. Vrank: A context-aware approach to vulnerability scoring and ranking in SOA. In *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*, pages 61–70. IEEE, 2012.
[17] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.
[18] Damjan Kovač and Denis Trček. Qualitative trust modeling in SOA. *Journal of Systems Architecture*, 55(4):255–263, 2009.
[19] Yutu Liu, Anne H Ngu, and Liang Z Zeng. QoS computation and policing in dynamic web service selection. In *Proceedings of the 13th international World Wide Web conference*, pages 66–73. ACM, 2004.
[20] Lutz Lowis and Rafael Accorsi. Vulnerability analysis in SOA-based business processes. *Services Computing, IEEE Transactions on*, 4(3):230–242, 2011.
[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. 1999.
[22] Chittoor V. Ramamoorthy, Gary S. Ho, and Yih-Wu Han. Fault tree analysis of computer systems. In *AFIPS National Computer Conference*, 1977.
[23] Florian Skopik, Daniel Schall, and Schahram Dustdar. Modeling and mining of dynamic trust in complex service-oriented systems. *Information Systems*, 35(7):735–757, 2010.
[24] Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, February 2005.
[25] Trusted Computing Group. Trusted platform module library specification, family 2.0, level 00, revision 00.99. Main specification, Trusted Computing Group, October 2013.
[26] Yao Wang and Julita Vassileva. Toward trust and reputation based web service selection: A survey, 2007.
[27] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. Heading off correlated failures through Independence-as-a-Service. In *11th OSDI*, August 2014.