An Independent Verification of Errors and Vulnerabilities in SaaS Cloud

Rajeshwari Ganesan
Infosys Labs, Bangalore India
Email: rajeshwari_ganesan@infosys.com

Santonu Sarkar
Infosys Labs, Bangalore India
Email: santonu_sarkar01@infosys.com

Naveen Tewari
Infosys Labs, Bangalore India
Email: naveen_tewari@infosys.com

***Abstract*—Software-as-a-Service (SaaS) offers immense advantages to a subscriber, as the SaaS subscriber pays for the amount of service he has consumed. This pay-per-use model offered by the SaaS provider is supported by the underlying virtualization technology, which allows sharing of the physical infrastructure among several clients who subscribe to the SaaS cloud to optimize the cost of usage. However, with this cost-benefit come several risks related to reliability, security and availability (RAS). Consequently, a potential subscriber of a SaaS offering wants to perform several reviews and validations related to RAS before the subscription. In fact, the subscriber often prefers an independent validation of these QoS aspects, as an on-going basis. In this work, we propose a validation methodology and a tool iCirrus-Val for a SaaS subscriber, to perform an independent analysis and verification of functional errors and vulnerabilities of a SaaS cloud from its weblogs. iCirrus-Val groups the logged URLs into whitelist and suspect categories. A whitelisted URL belonging to a business process, is analyzed for permanent and transient faults. A suspect URL is further analyzed to check if it falls into an "attempt to an attack category".**

Our approach is lightweight and does not require data from other parts of the system that is typically unavailable to a SaaS subscriber. It is restricted to a study of RAS from the subscribers entry point. However, we believe that our approach has a potential to identify a large number of the vulnerabilities. Our belief, though not empirically validated here, rests upon recent research findings which indicate that vulnerabilities are increasingly targeted at the entry point such as the web server, as attackers find it difficult to hack the core server. We illustrate our approach using a real-life data of an organization that has adopted a SaaS public cloud.

***Keywords*-permanent and transient error; vulnerability; SaaS;**

I. Introduction

A Software-as-a-Service Cloud (SaaS cloud) offers immense advantages to a subscriber as the management and maintenance of the IT systems are delegated to the cloud service provider (CSP). An example of a highly successful Public SaaS cloud is Salesforce.com[1] that provides a customer relationship management (CRM)[2] product as a SaaS. A subscriber to such a SaaS solution saves on initial upfront cost of IT and pays the provider (i.e. "Salesforce.com"), based on the usage of the service. The underlying technology backbone for such a cloud infrastructure is the virtualization technology that offers elasticity, scalability and redundancy to meet workload demands. Using virtualization, the same physical infrastructure is shared among several subscribers of the public SaaS cloud, thereby optimizing the cost of usage. However, along with these benefits, several risks related to reliability, availability and security (RAS) are also associated with a public cloud based solution. These risks manifest due to sharing of physical infrastructure, usage of virtualization software and exposure to the Internet. These risks have a negative influence on the trust factor of the solution.

In the best interest of the CSP, it is necessary to have a trusted solution. Given the service delivery model, the client using the public cloud can discontinue the service at the onset of SLA violations. This could also negatively influence other subscribers of the service. An organization's adoption to a specific public cloud service begins with an understanding of reviews from other customers, research reports, or other documentation. The CSP compliance to relevant standards such as SAS70[3] or ISO27002:2005[4] reduces the resistance of adoption. However this alone is insufficient; the organization often asks for a "request for proposal" (RFP) from the CSP to evaluate the the architecture, deployment, security framework and understand how the CSP can ensure the RAS of the service. Even after moving to the public cloud, many subscribers plan for an independent verification of the risk at a periodic interval for a continuous risk mitigation.

In this paper, we propose a methodology to analyze RAS from a subscriber perspective by analyzing the subscriber web access logs. We analyze various permanent, and transient errors to understand the reliability. Along with this, the study of response HTTP status code helps us to determine the availability of the SaaS platform. To evaluate the system vulnerability, we examine suspect URLs to determine "attempts to attack". We have implemented this methodology in a tool called iCirrus-Val[5]. We further illustrate our approach using a real life case-study of a subscriber organization that adopted a public SaaS cloud, and present the details of a preliminary independent verification carried out using iCirrus-

[1] http://www.salesforce.com/in/crm/products.jsp

[2] http://en.wikipedia.org/wiki/Customer_relationship_management

[3] SAS stands for statement on auditing standard (http://sas70.com)

[4] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50297

[5] iCirrus-Val is a part of Infosys Lab's Cloud QoS framework called iCirrus

978-1-4673-2266-9/12/$31.00 ©2012 IEEE

Val during the early deployment phase of the subscriber.

The paper has been organized as follows: Section II describes the client's context and background with the business drivers that led to the adoption of the cloud, architectural details of the private cloud, and its security framework. Section III describes the approach we took to meet the objectives and goals of the exercise (how safe is the customer's public cloud deployment). Section IV presents our observations and conclusions. In Section V we describe the related work. In Section VI, we conclude the paper.

II. Case Study: A Public SAAS Cloud

We introduce a subscriber which we have studied for fault identification and vulnerability detection. The subscriber is an organization in retail domain, with more than thousand stores spread across USA and Canada. The organization had inefficiencies in the existing IT system and there was a lack of process standardization in their contract management, book keeping, employee management and payroll processing across different offices. Rather than investing further to fix these issues, the organization chose to move to a public SaaS cloud. The SaaS platform on the public cloud is based on PeopleSoft[6] and it caters to multiple organizational subscribers as shown in Figure 1. A subscribing organization can choose a set of modules such as *contracts, billing, general ledger, mobile time and expenses, payroll processing*, which are based on PeopleSoft's Service Automation subsystem. A complete reference of these functional modules is publicly available at [1], [2]. In our case study, the SaaS based solution was adopted organization-wide, and used by the employees as well as the contractors spread over the countries. While the strategy helped the subscriber to standardize the processes without any upfront investment in IT, the subscriber decided to undertake an early independent evaluation of the RAS risks involved in moving to such a public SaaS cloud. In the next section, we describe this evaluation methodology and the outcome of our verification.

A. Functional Architecture

As shown in Figure 1, an employee of a subscriber organization connects to the SaaS platform through a secured HTTP connection in order to access different services, as mentioned earlier. It is possible to perform administration of the SaaS instance specific to the subscriber using the browser based interface. For illustration, let us consider the "payroll processing" service for employees and contractors of the organization. The service can compute the gross net earnings, deductions, associated taxes and levies. For this organization, the most crucial part was the implementation of country specific taxation rules. The underlying platform used standard templates and used data stored in the database to generate dynamic documents such as online paychecks in PDF format. For the sake of brevity we don't illustrate other business use cases.

[6]http://www.oracle.com/in/products/applications/peoplesoft-enterprise/index.html

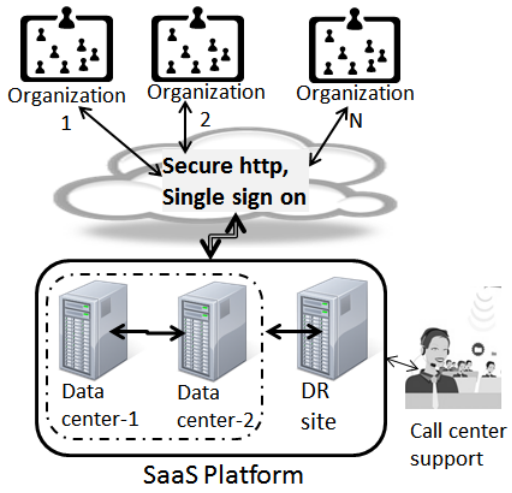

Figure 1. Schematic Diagram of the business platform

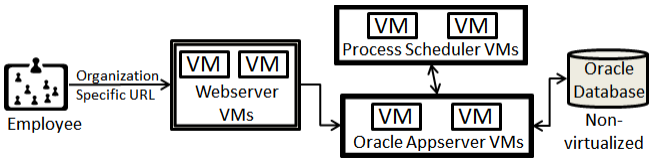

Figure 2. Flow of an employee request

We now illustrate a simplified user request flow for this platform, depicted in Figure 2.

1) An user (employee or a contractor) logs into the SaaS platform using the subscriber specific URL.
2) The request from the user is processed by a cluster of web server virtual machines (VMs).
3) The request then flows to the application server which retrieves data and objects from the common database.
4) In addition, the process scheduler jobs are executed frequently on dedicated VMs forming the process schedulers.

B. Architectural Decisions

The SaaS platform provider has taken several architectural decisions for the implementation and deployment of the platform. While the web servers are deployed in a Windows 2008 OS, the IBM Websphere application server instances run in AIX 6.1 OS. The application server hosts the J2EE based Oracle PeopleSoft. Furthermore, all the objects (such as pages, menus, customizations) and data are stored as tables. Thus, the database is the most crucial for trust in this framework. As shown in Figure 1, the platform runs on two data centers and a disaster recovery (DR) site.

1) Virtual Machine Management: For the subscriber under study, the CSP assigned two VMs to host web servers, two VMs for the application servers and two VMs for process

schedulers. Two VMs for web server as well as application server perform load balancing of user requests. In order to ensure *higher availability* and avoid single point of failure, VMs pertaining to a given subscriber are distributed across different physical boxes which are virtualized. The database, however, is not virtualized.

2) Security Management: The SaaS provider incorporated several security best practices that comes with Peoplesoft platform [3]. Though these best practices were proposed keeping a traditional non-cloud deployments with dedicated machines in mind, the provider adopted them in the context of public cloud. For authentication and session management, the provider has taken the following design decisions:

Authentication: Authentication and authorization models use HTTPS and WS-Security. A lightweight directory access protocol (LDAP) v3 based authentication has been used for single sign-on and authentication.

Session ID based attack prevention: The CSP decided to use a pre-built and well tested client and server session management component, as opposed to a home-grown session management, since the home grown solution could be prone to security vulnerabilities. The pre-built solution has the session timeouts and client session state expiration policies to protect the system against many session ID based attacks.

Phishing attack prevention: The CSP decided to restrict access to the website to known set or *whitelisted* set of URLs. For example, even if a customer with legitimate credentials logs from a network not registered with the CSP, his access will be disabled. This decision was taken to reduce phishing attacks.

Penetration test: The CSP created a subscriber specific penetration testing strategy to prevent the system from known exploits and attack techniques. For the subscriber in this case study, the test suite included checking for over 35 known vulnerabilities covering LDAP injection, SQL injection, cross-site scripting, web site vulnerabilities and so on. The scores indicated that severity of these vulnerabilities were low. The CSP shared these scores to the subscriber periodically.

C. Independent Verification

Despite the above mentioned strategies and periodic reports on the security issues from the CSP, the subscriber required an independent verification of RAS risk requirement. When we performed the verification from the subscriber point of view, we may not always have the complete access to the infrastructure details of the cloud provider, such as the details of the physical machine configuration, the software used, VMs utilization statistics, the underlying network topology, the other tenants co-hosted with the subscriber under consideration and so on. While a comprehensive independent verification would require aforesaid information, our verification remains restricted to the SaaS system entry point for a particular subscriber.

date	time	time taken	bytes	c-ip	sc-status	cs-method	cs-uri
2012-03-01	06:02:50	0.015	13799	10.x.y.z	200	POST	/psp/BW/?cmd=login

Table I
SAMPLE WEBLOG STRUCTURE

Entry point based vulnerability analysis, however, is not that ineffective as indicated in a recent research finding [4]. This finding shows that vulnerabilities are increasingly targeted towards the entry-point, such as the web server, as attackers find hacking in core server more and more difficult. Hence, our strategy can be considered to be a good starting point. The next section describes the approach we took for the verification.

III. VERIFICATION APPROACH BASED ON WEBLOG ANALYSIS

We have built a tool iCirrus-Val to perform an off-line analysis of historic user access logs that are stored in each of the web servers as HTTP access logs containing every single HTTP request made to the web server over a period of time. Analysis of access logs for various application quality of service has been an important focus area [5], [6] for a long time. In most of these cases, the attention was given towards application performance. In case of iCirrus-Val, the access log is the source for RAS analysis. Here, we have used an open source WTOP tool[7] that could amongst several things sieve legitimate user requests from suspects, determine errors and unavailability issues. The current version of iCirrus-Val uses Perl scripts to perform access log data analysis.

A sample access log format is shown in Table I. It contains time stamp, URL, time-taken to process the request, HTTP status codes, and IP address of the client requesting the URL. Since our objective is to determine site errors and unavailability issues, the 3 digit HTTP status codes [8] are most useful to us. Table II lists HTTP codes that we are interested to analyze further from the weblogs. For instance, though the error code 404 would mean "page not found", we use this information in conjunction with URLs and time information to indicate permanent, or transient errors, or even a potential vulnerability.

A. URL Categorization

iCirrus-Val follows the steps below to create a URL categorization as shown in Figure 3:

1: *Classify Whitelisted and Suspect URLs:* We collect the whitelisted URLs as a set of regular expressions that represents the valid URLs implementing the various business scenarios. While analyzing the weblog, when iCirrus-Val

[7]http://code.google.com/p/wtop/

[8]For details one may refer to the original RFC at http://www.w3.org/Protocols/rfc2616/rfc2616.html

Status code	Our interpretation
200-success	Determine malicious attacks that were successful
403-forbidden	Determine insufficient permission resulting in access violation
404-not Found	Find permanent, or transient errors in Whitelisted URLs
405-method not Allowed	Determine errors triggered by user ad-hoc query or functional errors
500-server Side Error	Run time failures due to unhandled exceptions in the white listed URLs

Table II
HTTP STATUS CODES AND THEIR USE FOR ERROR/FAILURE DETECTION

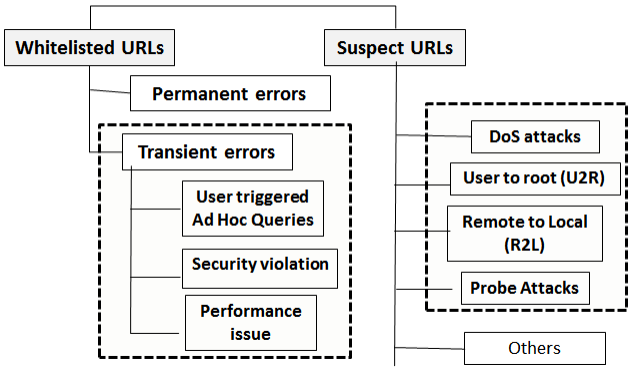

Figure 3. URL Categorization

finds that a URL is not in the supplied whitelist, it marks the URL as "*Suspect*" to be probed for further verification. This Suspect list could have malicious scripts, arbitrary files or attacks. iCirrus-Val maps the Suspect URLs conforming to an attack into categories of (a) Denial of Service attacks (b) User to Root (U2R) attacks, (c) Remote to Local (R2L) attacks, and (d) Probe attacks, proposed by Kendall [7].

2: *Determine Permanent Errors:* Functional bugs in the software that are not trapped during software testing phase results in errors. Irrespective of the run time conditions they always result in errors. The HTTP status code always returns an error code when the specific URL with a bug is accessed.

3: *Determine Transient or Intermittent Errors:* Often run-time conditions influence the success of a user request. One typical example is server overload that causes deadlocks, and unavailability of a resource due to timeouts. While it is possible to identify transient errors, determining the root cause of these failures can be difficult. For example, to see if server overload was the cause, we would need system resource utilization statistics. Here, we extract the transient errors by finding URLs with success HTTP status code sometimes and status codes (≥ 400) some other times.

IV. RESULTS AND DISCUSSION

We have run iCirrus-Val on web access logs collected for two months at the two web servers of the subscriber. There were 70 log files, where each file corresponds to a log entry for a day. We have consolidated these log files, taken a fixed time interval of one hour and calculated the number of log entries in each time interval.

A. Whitelisted URLs

iCirrus-Val identified close to 60000 hits that belongs to whitelisted URLs with status code ≥ 400 (i.e. error) from these log files. Figure 4 shows a distribution of these errors for two months (The X-axis represents sampled time intervals). Out of these errors, 11% were permanent errors (6200) while 89% were transient errors as shown in Figure 5a.

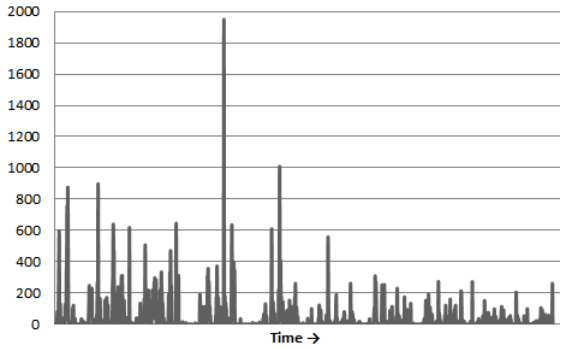

Figure 4. Number of whitelisted URL hits (hourly basis) that resulted in an error

1) Permanent Errors: There were 200 whitelisted URLs that caused permanent errors. On manual investigation of these permanent errors, we observed the following:

1) Several static PDFs and documents were not found and they were also infrequently accessed. It was also observed that many of these errors were subsequently fixed in the following month. Around 20% of those remained.
2) There were several static page level objects such as images, icons missing in several pages.
3) Many requests were internal server errors, with HTTP status code ≥ 500 that related to cache. The exact reason could not be ascertained and further analysis to determine root cause was to be undertaken by the CSP.

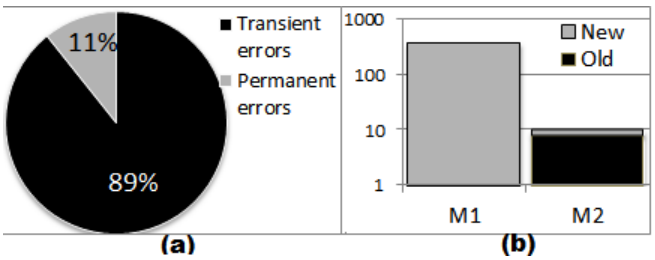

Figure 5. Propagation of errors from month to month

iCirrus-Val also calculated how many of these permanent errors were carried forward to the next month as shown in the histogram in Figure 5b. The bar "M1" indicates the number of errors in the first month M1, that are new. For the next month M2, represented by the second bar, the lower part of the stacked bar indicates the number of errors that were carried from the previous month (in black color). We have observed that only 2% of the errors from "M1" was carried to the next month M2. Additionally a few new errors, shown in gray colored slab, were introduced in the next month. The Y-axis is in a logarithmic scale of base 10.

2) Transient Errors: Over 285 URLs contributed to the 89% transient errors. The following are our key observations:

Access forbidden: While loading of images, CSS and Javascripts, there were few 403 errors indicating that access to these resource are forbidden.

Dynamic queries: In several cases dynamically created PDFs were not found. The 3.5% of the total dynamically loaded PDFs, belonging to payroll processing and benefit enrollment process category had this error.

Business function: Many business functions also had transient errors resulting in server side errors (HTTP code 500). These were primarily caused by ad-hoc queries formed and fired by users. We have encountered 21% of the ad-hoc queries resulted in errors.

Impact on response time: Most of these ad-hoc queries had very high response time, exceeding 15 minutes, indicating that they could have a potential impact on the system performance. It would be interesting to see the number of errors that occur due to this performance degradation when these ad-hoc queries are fired.

B. Suspect URLs

iCirrus-Val identified 1200 Suspect URLs during the two months observation period. Over 99% of these were determined to be "attempt to attack" by iCirrus-Val, using a custom-built database of known vulnerabilities and patterns, based on [7], [8]. A few patterns used are listed in Table III. Using this initial classification, a

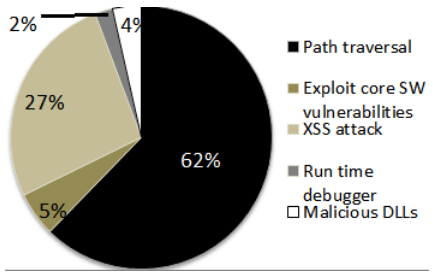

Figure 6. Distribution of attack types

manual verification was done to list interesting findings as this vulnerability detection process can never be fully automated. iCirrus-Val classified these errors to a set of

Pattern sample	Possible vulnerability
/../../../../../<filename>	Path traversal
<script> </script> xss cross_site_scripting.nasl	XSS and header injection
http-equiv=Set-Cookie document.cookie=	Cookie vulnerability
JavaSnoop	Runtime debugger
/servlet/com.newatlanta. servletexec.JSP10 Servletfsmsh.dll?	DOS attack
/system/autoexec.ncf	Password stealing

Table III
SAMPLE PATTERNS USED BY ICIRRUS-VAL FOR VULNERABILITY DETECTION

well-known root causes [8]. We found that most of these "attempts to attacks" resulted in 404, indicating that they failed. The following are list of the attacks:

Path traversal: A number of attempts for performing several path traversals primarily to determine password files and configuration files. There were over 228 requests of this category.

Known product vulnerability: Several requests being made to exploit known software vulnerabilities of the constituting software. Of these 20 attempts were made on ColdFusion, a product used here for developing and deploying web applications. Again related to viewing of file contents.

Script injection: There were 97 indicated attempts to cross site scripting attacks such as execution of an ASP program invoking a Javascript with a request string.

OS level injection: 8 attempts were made to use runtime debugger to the Java process. If successful, these could get access to function names and parameters.

Cookie vulnerability: There were 13 attempts to run dynamic link libraries (DLLs). Most of them were intended to get cookie related information either from the web server using HTTP response header, or from a stored document.

The others were attempts to Denial of Service attacks, Registry corruption, Password stealing, attempts to target administrators client state and authentication details to access or take actions as administrator.

1) Observation: One of the key vulnerability exploited by hackers is the understanding of the file names that would enable them to access, perform de-compilation, and do a code examination to determine vulnerabilities. The URLs used by the subscriber are publicly known, which could have been a source of exploitation by the attackers. The duration of the longest attack was close to an hour, with no pre-measures to raise alarm.

The web server VMs were load balanced in which each HTTP request could be serviced in any of the web server. However, we observed that during the period of attempt to attack, all the requests were always going through one web server. So the attacker had control over this web server.

It appears that the specific hardware and software component of the SaaS environment are known to the attackers. For example, we found attempts to execute arbitrary code on the installed hardware specific network node manager. Similarly, it appears that the underlying software infrastructure (Java, Coldfusion) were known to the attackers.

V. Related Work

Detection and prevention of vulnerabilities has been an active field of research for more than a decade. Intrusion detection and response is an important component of vulnerability assessment and prevention. While DARPA created a corpus to computer intrusion detection evaluation in 1998, the work by Kendall [7] defined a classification framework for various attacks, including ones related to a virtual machine. Automated intrusion response strategies have gained momentum in recent times [9], [10], where these strategies assume that the adversary pretends to be a user in order to attack the system. Another class of attacks, reported in [11] describes how one can exploit a hypervisor weakness in a shared infrastructure like Amazon EC2. In this highly cited work, authors describe how a malicious VM becomes co-resident with another VM and exploits the VM side-channel weakness to extract information of a collocated VM. An early work by Lombardi et al.[12] proposes a protection system for VMs based on monitoring. More recently, the work by [13] has focused on EC2 to show possible ways to extract information from collocated VMs, `ssh` vulnerabilities, and a few remedial measures. The work by Sekar et al.[14] proposes an accounting mechanism for resource consumption, from the perspective of whether a tenant has really consumed what he has charged. Thus, this work can be treated as a quantitative measurement of trust of the shared infrastructure service provider. A work by [15] reviews six categories of security issues related to communication and infrastructure. Though most of these works have the public cloud in mind, the security issues are applicable in a private cloud scenarios as well. Our work can be considered as a light-weight, non-intrusive assessment approach which relies on the weblogs to perform a post-facto analysis of vulnerabilities.

VI. Conclusion

In this paper we have proposed a methodology and a tool iCirrus-Val to analyze permanent and transient errors as well as potential vulnerabilities for a SaaS subscriber by analyzing the web access logs of a real-life SaaS platform. iCirrus-Val found several potential vulnerabilities, as well as permanent and transient errors. We observed that a small percentage of permanent errors were carried from one month to the next.

Our methodology, though effective, has a few limitations. Our approach does not attempt to find how the attack was originated. We have not analyzed any temporal correlation of suspect entries. Beyond identification, we have not categorized the attacks into different severities.

With regard to the future work, we plan to include several other vulnerability signatures in our database. Our current approach of detecting vulnerabilities are based on a pattern matching. We would like to analyze the temporal patterns and employ machine learning techniques in the future. We would like to investigate whether a transient error is correlated with the response time degradation. Our approach currently uses only web access logs. We intend to use other data such as utilization statistics, to improve the detection capability.

Acknowledgment

The authors would like to thank Prof Zbigniew Kalbarczyk and Prof Ravi Iyer of the University of Illinois at Urbana Champaign for their valuable feedbacks during various stages of our experiments and paper writing.

References

[1] J. Doolittle, *PeopleSoft Developers Guide for PeopleTools & PeopleCode*, Oracle, 2008.

[2] J. J. Marion, *PeopleSoft PeopleTools Tips & Techniques*. Oracle Press, 2010.

[3] D. Kurtz, *PeopleSoft for the Oracle DBA*. Academic Press, 2012.

[4] C. McNab, *Network Security Assessment*, 2nd ed. O'Reilly Media, Inc, 2011.

[5] F. Facca and P. Lanzi, "Mining Interesting Knowledge from Weblogs," *Data & Knowledge Engineering*, vol. 53, pp. 225–241, 2005.

[6] L. Borzemski, *Measuring of Web Performance as Perceived by End-Users*. Hershey: IGI Global, 2008.

[7] K. Kendall, "A database of computer attacks for the evaluation of intrusion detection systems," Master's thesis, Massachusetts Institute Of Technology, 1999.

[8] D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook*. John Wiley & Sons, 2011.

[9] A. Sharma, Z. Kalbarczyk, R. Iyer, and J. Barlow, "Analysis of Credentials Stealing Attacks in an Open Networked Environment," in *NSS*, 2010.

[10] B. Foo, Y. Wu, Y. Mao, S. Bagchi, and E. Spafford, "Adepts: Adaptive intrusion response using attack graphs in an e-commerce environment," in *DSN*, 2005, pp. 508–517.

[11] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," in *ACM CCS*, 2009, pp. 199–212.

[12] F. Lombardi and R. D. Pietro, "Transparent Security for Cloud," in *SAC*, 2010, pp. 414–415.

[13] S. Bugiel, S. Nrnberger, T. Poppelmanny, A.-R. Sadeghi, and T. Schneider, "AmazonIA: When Elasticity Snaps Back," in *ACM CCS*, 2011, pp. 389–400.

[14] V. Sekar and P. Maniatis, "Verifiable Resource Accounting for Cloud Computing Services," in *ACM CCSW*, 2011, pp. 21–26.

[15] J. C. Roberts-II and W. Al-Hamdani, "Who Can You Trust in the Cloud?-A Review of Security Issues Within Cloud Computing," in *Information Security Curriculum Development Conference*, 2011, pp. 15–19.