http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 13 (14): 2240-2249, 2014
ISSN 1812-5638 / DOI: 10.3923/itj.2014.2240.2249

Evaluation Dependability Attributes of Web Application using Vulnerability Assessments Tools

Hasan Kahtan, Nordin Abu Bakar, Rosmawati Nordin and Mansoor Abdullateef Abdulgabber
Faculty of Computer and Mathematical Sciences, UiTM Shah Alam, Selangor, Malaysia

Abstract: Current organizational vulnerabilities primarily originate from web applications. Security holes in web applications have led to credit card theft and damage to the financial resources of institutions. Existing literature shows that dependability attributes are the solution for security vulnerabilities in web applications. However, efforts to measure dependability attributes remain under investigation. This study aims to measure the vulnerabilities that affect the dependability attributes by Vulnerability Assessment Tools (VATs). VATs are utilized to identify vulnerabilities in current web application systems. The assessment is performed based on six dependability attributes: Availability, reliability, confidentiality, integrity, safety and maintainability. The assessment provides methods and strategies that solve the lack of measuring dependability attributes of web applications. The assessment results will also realize the awareness of software development industries on the effect of neglecting dependability attributes in the software development process.

Key words: Web application, dependability attributes, availability, reliability, integrity, confidentiality, safety, maintainability, vulnerability assessment tools

INTRODUCTION

Software applications are essential in today's society as they runs the machines that help people live smoothly. Software applications can be found in most items that are used in daily life, such as cars, cell phones and kitchen appliances. As a result, the competition is increasing among companies specializing in software production and services. Companies compete even on trivial matters with the goal of producing software applications that are dependable, reliable and affordable. A software application must be developed more efficiently and securely to achieve these goals. The security of these software applications or software security, has thus become the primary concern in their evolution. Software security is engineering software that can continue to function correctly under malicious attacks (McGraw, 2011). Software security involves helping software developers perform a good job to remove the burden of security problems from end users.

Software security requires the incorporation of a secure code during the development stage to avoid potential vulnerabilities. The organizational software development life cycle includes vulnerabilities assessment, application scanning and penetration testing. The research community still experiences difficulties in creating a scale that can measure the security level of a software system. Software security is a challenging issue because it is an elusive problem that should be resolved in all software development stages and at different levels of conceptualization (Mir and Quadri, 2012). The increasing number of attacks trouble business owners because many aspects of business enterprises are integrated with the internet. These attacks can have various influences on companies, such as Denial of Service (DoS), data loss, loss of productivity and liability claims. An entire industry that addresses computer security also arose from the severity of such attacks (Singh, 2012).

Today, network firewalls and vulnerability scanners are commonly used for web application security. Firewalls prevent unauthorized persons from accessing systems and vulnerability scanners check systems for possible weaknesses. However, neither of these systems can guarantee that different kinds of attacks can be identified. Many studies have stated that the only way to solve software vulnerability is to consider software security development (Han and Khan, 2006; Kim, 2004; McGraw, 2004; Mir and Quadri, 2012; Simpson, 2012). However, efforts to measure software security remain under investigation (Alberts *et al*., 2012; Colombo *et al*., 2012; Karen *et al*., 2006; Lai, 2012; Steward *et al*., 2012). This condition is due to the need to specify the security attributes' requirements during the early stages of the system development and the complex nature of the operational environment itself. In fact, the development of

Corresponding Author: Hasan Kahtan, Faculty of Computer and Mathematical Sciences, UiTM Shah Alam, Selangor, Malaysia

a well-established scale that can measure the security of a software system remains a great challenge for the research community (Cinque *et al*., 2010).

Vulnerability Assessment Tools (VATs) is a significant alternative that can be used to protect against common security problems measures (EC-Council, 2010). Operation handlers should carefully examine security breaks in fielded systems. Attacks occur despite powerful design and implementation. Monitoring the behavior of software is thus an excellent defensive technique. The knowledge acquired through studying attacks should be brought back to development organization. Security practitioners should also openly monitor vulnerability models and attack patterns. They should also take note of risks that emerge throughout all stages of the software life cycle to facilitate the creation of a constant risk analysis thread which includes the tracking of recurring risks and the monitoring of activities (Hartel, 2012).

This study is the extension of the previous study in Kahtan *et al*. (2012, 2014a, b). Kahtan *et al*. (2012) addressed the challenges of security features in Component-Based Software Development CBSD models. Kahtan *et al*. (2014a) presents the results of a survey pertaining to the embedding of security features in the CBSD process. The survey results showed that the embedding of security features in the software lifecycle is crucial because the incorporation of security activities in CBSD will minimize vulnerabilities in the software system, thus reducing system cost. Kahtan *et al*. (2014b) proposed the software security approach to increase the security in CBSD. Six dependability attributes (i.e., availability, reliability, confidentiality, integrity, safety and maintainability) are highlighted as widely accepted attributes to achieve better security in most software components. The current study aims to address the two main issues of measuring the level of security (or dependability) in software components and identifying the tools used to measure software vulnerabilities. The objective is to investigate the effect of software development without considering security (or dependability) attributes by using VATs. Meanwhile, VATs detect the vulnerabilities that can cause system failure, suspension or denial of service that can affect the dependability attributes.

VULNERABILITY ASSESSMENT TOOLS (VATS)

VATs identify weaknesses that are present in a system, thereby improving or changing the security posture of networks or systems (Wales, 2003). VATs involve compromising servers, web applications, wireless networks and other exposed locations so that security issues can be located and analyzed (Manoj *et al*., 2002). VATs is a systematic evaluation of networks used to recognize security defiance and determine the appropriate security measures (EC-Council, 2010). Such tools also come in the form of security scanners that have a protective approach to network security. VATs are used to identify security vulnerabilities and enable stakeholders to solve vulnerability problems, hence, significantly increasing security protection. The VATs method strengthens the system according to the identified issues and helps prevent the likelihood of attacks in the future. The objective of VATs is to deliver efficient, thorough and automated identification to detect known vulnerabilities in the configuration of a specific operating system (Manoj *et al*., 2002; Planquart, 2001). VATs are vital because vulnerabilities are exploited by malicious software or attackers; they also map known weaknesses in the network and provide an expert assessment of such potential weaknesses (EC-Council, 2010).

VATs function as documented networks or make up a system security defects database. They also try to provide severity categorization in the final reports and inspect every defect in the available services of the target range of hosts. VATs are processes of locating, calculating and classifying vulnerabilities in a system. Web applications and networks are analyzed to identify the occurrence of vulnerabilities. These vulnerabilities occur because of incorrect software design, unsecure authentication or misconfiguration. VATs collate known vulnerabilities in the system and present an assessment of potential vulnerabilities that can be exploited by malicious software or attackers. VATs can also be used to determine the following: (a) System tolerance to attack patterns, (b) Additional methods that can mitigate system vulnerabilities and (c) Ability of defenders to detect and respond to attacks.

RELATED WORK

Current organizational vulnerabilities primarily originate from web applications. Security holes in web applications have led to credit card theft, damage to the financial resources and reputation of institutions and compromised computers (Secunia, 2012; Sophos, 2012; Verizon, 2012). Vulnerabilities enable external exploitation and system penetration, thus leading to unauthorized access and compromising the dependability of systems or web applications. Vulnerabilities are system weaknesses that are considered inherited defects in systems or web applications. Vulnerabilities are also the doorways that reveal threats (Umrao *et al*., 2012).

In literature, numerous books and articles have been written from the security tools' perspective. Included in these studies are discussions on security testing strategy usage, switches and techniques. In the present study, the focus is on security testing using VATs. Several examples of related works on VATs are discussed in brief.

Espadas *et al.* (2013), for example, used the Apache JMeter tool to create workload and perform availability measurement. The workload is created to the Tomcat cluster installed in a cloud environment where the SaaS platform has been deployed. In Gao *et al.* (2012), the authors proposed a testing as-service (TaaS) infrastructure and reports a cloud-based TaaS environment with tools (or CTaaS) developed to meet the needs in SaaS testing, performance and availability evaluation. The CTaaS feature allows users to select the available load test script set and its corresponding test tool, such as JMeter.

Brokken (2013) discussed the available open source tools to maintain and monitor the integrity of the individual computer systems and the network organization. The authors highlighted that open source vulnerability scanner (OpenVAS) can be used to improve and monitor the integrity of an organization's IT infrastructure. A study conducted by Morris *et al.* (2011) used OpenVAS to perform port scans of the tested phasor measurement units and phasor data concentrators. OpenVAS attempt to device security feature identification, port scans and network congestion testing.

Cowan (2003) reviewed several software auditing tools. The objective of these auditing tools is to enhance the security and maintainability of open-source systems. The authors highlighted that Rough Auditing Tool for Security (RATS) can help developers to prevent vulnerabilities by auditing software for flaws from the start. Curphey and Arawo (2006) described the different technology types to analyze web applications and web services for security and maintainability vulnerabilities, along with each type's advantages and disadvantages. The authors highlighted that RATS uses syntactic and semantic inspection of programs to find vulnerabilities in software codes. Tevis and Hamilton (2004) also provided a brief description of the available source code security auditing to partially automate the security analysis of software. The authors highlighted that RATS uses greedy pattern matching to find potential errors. Consequently, false positives are prone to occur more often.

In general, several vulnerability assessment tools exist. However, this study primarily focuses on three VATs that measure the vulnerabilities affecting dependability attributes. Table 1 summarizes the usage of VATs in the related discussed literature concerning dependability attributes.

VATs METHODOLOGY

In the present study, a vulnerability assessment was conducted to prove the existence of vulnerabilities in web application systems. This assessment was performed to investigate the effect of a poor software development system developed without embedding the dependability attributes. VATs were used to identify vulnerabilities in the current web application systems where a total of 210 web application systems were chosen in random for this assessment. The assessment was performed by the VATs method on six dependability attributes: Availability, reliability, confidentiality, integrity, safety and maintainability. Hence, VATs were used for the web application assessment. The assessment tools include Apache JMeter, OpenVAS and RATS. The VATs methodology was divided into the assessment process and the assessment tools and setup.

Process: A vulnerability assessment process outlined five phases: Planning phase, vulnerability mapping phase, attack execution phase and documentation and reporting phase.

- **Planning phase:** Information for assessment execution, such as assets, vulnerabilities of interest against assets, dependability controls for the mitigation of vulnerabilities and development of the assessment approach, was collected in this phase
- **Vulnerability mapping phase:** Sufficient information was gathered on the targeted web application from the previous phase. The vulnerability mapping phase identifies and analyzes vulnerabilities based on disclosed ports, services and dependability attributes

Table 1: VATs pertaining to dependability attributes

Tools	Dependability attribute	References
Apache Jmeter: An open source software Java desktop application that is designed to load test functional behavior and measure performance. It can be used to simulate a heavy concurrent load on a J2EE application and analyze overall performance under various load types	Availability Reliability	Espadas *et al.* (2013), Gao *et al.* (2012) Apache JMeter (2010)
OpenVAS: An open source vulnerability scanner that scans networkes for vulnerabilities and creats a report based on network status	Integrity, Confidentiality, Safety	Brokken (2013), Morris *et al.* (2011), OpenVAS (2013)
Rough Auditing Tool for Security (RATS): A security-auditing utility for a variety of languages	Maintainability	Cowan (2003), Curphey and Arawo (2006), RATS (2013), Tevis and Hamilton (2004)

to provide the auditor a clear vision to carefully examine any known or unknown vulnerability. Vulnerabilities that affect dependability attributes were determined in this phase, the obtained inputs are the different elements found during the information gathering process

- **Attack execution phase:** The targets selected in the previous step can be attacked by the discovered vulnerabilities. The attack execution phase may require additional research or modifications to work properly. This phase aims to identify vulnerabilities and address the activities associated with the intended assessment method and technique
- **Documentation and reporting phase:** The testing process was concluded by recording, illustrating and presenting the vulnerabilities found. Types of reports created for each relevant authority at the contracting organization may have different outlooks to understand and analyze the security breaches in their IT infrastructure. These reports can thus fulfill the purpose of vulnerability assessment by capturing and comparing the target system with the dependability attributes

Assessment tools and setup: The tools involved in each of the assessment steps are discussed in this subsection. The setup steps are also shown in brief.

- **Apache JMeter setup:** In this study, JMeter was configured to create distributed requests to simulate workload usage. JMeter has a simple and intuitive user interface; thus, the tool can be utilized after a short learning period. The website is stable and has a strong online community so that a user can expect free expert solutions to problems when using the tool. The selected web applications in this study contain several requests from different hosts that implement JMeter tests. Please see the user manual provided by (jmeter.apache.org) for the details on Apache JMeter setup and configuration

After the setup and configuration of the test plan, the workload for the web application system was generated. JMeter plugins provide an extensive set of results and thus allow the evaluation of system behavior from the user's perspective. The requests were sent to the web application system.

Workload was generated synthetically by executing a number of threads that emulate users accessing the application being tested to systematically obtain workload-dependent performance data from web-based applications. Workload was generated with Apache Jmeter v2.9 containing Jmeter plugins v1.1.1 (jmeter-plugins.org), a set of JMeter extensions that allow the definition of custom request distributions and the representation of results in a graphical manner. A common approach is to replicate a number of pre-recorded traces into multiple executing threads. A user must build a test plan which is essentially a sequence of operations that JMeter will execute, to create a load test in JMeter. The test plan includes the following:

- **Thread group:** The elements in this group are utilized to specify a number of running threads (users) and a ramp-up period. Each thread simulates a user and the ramp-up period specifies the time to create the threads
- **Samplers:** These elements are configurable requests to the server HTTP, FTP or LDAP requests. The load test focuses on web service requests only
- **Listeners:** These elements are employed to post process request data. For example, data can be saved to a file and results can be illustrated with a chart. A JMeter chart does not provide many configuration options; however it is extensible and allows for the addition of an extra visualization or data processing module. Several plugins have been developed to overcome these deficiencies and make JMeter more useable for the tester community. For the load test in this study, the plugin Jp@gc bytes throughput over time was used. This plugin was employed to show the sent and received bytes over time and illustrate the results with a well-defined chart
- **Throughput:** Throughput is the total amount of requests divided by the time required to complete the experiment. A more detailed description of the configuration step of Apache JMeter along with the available elements is provided on the Apache JMeter web site

Jmeter was configured for the different tests to simulate 100, 150, 175, 200, 250, 300, 350 and 400 simultaneous users (threads). The ramp up period was set to 10 s, meaning the JMeter has 10 s to start all threads. For 10 threads, the JMeter starts one thread per second; for 100 threads, the JMeter starts 10 threads per second. JMeter was configured not to use HTTP keep-alive, indicating that every simulated user opens a new transmission control protocol socket that is closed immediately after the HTTP response is successfully received from the server. Table 2 presents the input criteria for load testing at particular seconds.

For each second, two runs of testing on both systems were performed. The results were saved as a comma-separated value file. The file was opened via Microsoft

Table 2: Input criteria for load testing at particular seconds

Time (Sec)	Thread No.	Loop	Ramp period
20	100	30	10
50	150	30	10
75	175	30	10
100	200	30	10
125	250	30	10
150	300	30	10
175	350	30	10
200	400	30	10

Table 3: JMeter readings results

Time (Sec)	Bytes
1	2808537
2	2288494
3	1368811
4	2726380
5	2278976
6	1259679
7	959679
8	53537
9	2980143
10	1891518

Excel to read the results obtained where a byte average for each second was displayed. Table 3 provides a tabulation of the results of JMeter reading for each second in 10 sec.

OpenVAS setup: In this study, OpenVAS 4.0.6 was installed by default on BackTrack 5 R3; no installation was hence required. However, OpenVAS 4.0.6 required configuration. The detailed steps of the OpenVAS 4.0.6 configuration can be obtained from OpenVAS (2013).

- **Starting OpenVAS:** OpenVAS operates in a client-to-server paradigm. Before scanning, a user must verify if the server is operating. The OpenVAS server can be initialized by executing the following commands:

```
sudo /etc/init.d/openvas-scanner start
sudo /etc/init.d/openvas-manager start
sudo /etc/init.d/openvas-administrator start
sudo gsad --http-only --listen=127.0.0.1 -p 9392
```

The listening ports in the machine were checked to determine if the OpenVAS server was operating. The OpenVAS server operates on port 9390. The command below was executed to verify if the OpenVAS server was initialized correctly.

```
netstat -ant | grep LISTEN
```

After checking that the server was operating, the client and vulnerability scan were initialized. The following command was executed to initialize the OpenVAS client:

```
Open http://localhost:9392/ or use shortcut
```

To log in to the site, the user name and password must be keyed. A target for scanning and a scan task were created after logging in. The button Target creation was clicked. The button Targets on the left hand menu was clicked and the term Web targeted was keyed in. In the Hosts dialog, the IP address of the server to be scanned was typed and the IP address information was inputted. The Name setting can be changed into a more descriptive form in the Comment dialog. The button Create Target was clicked upon completion.

Task creation: New task in the left hand menu was clicked. The scan should have a descriptive name. The scan name Web targeted was typed in. For Scan config, "Full and very deep ultimate" was selected. Scan targets was changed to the established target. The target Web Targeted was thus selected. Create Tasks was clicked to initialize the scan.

The next procedure was task initialization. The button Tasks in the left-hand menu can be clicked to view the configured scan Web Targeted. The green icon "play" was then pressed to begin the scan. Scanning a single host requires time and a much longer time is required to scan an entire network.

The results of the completed scan can be viewed by clicking the magnifying glass icon next to the scan in the Tasks menu. OpenVAS reports can be accessed by clicking the magnifying glass icon next icon to the completed scan in the Tasks menu. This procedure opens the Task summary window. Clicking the magnifying glass in the reports section of this window brings up the scan report. Results can be viewed by scrolling down the Report summary window. The report can be exported in a number of formats such as PDF.

From the scans, reports were generated that listed the vulnerabilities detected during the scan. Each report was the result of a security scan and contained the results of the executed plugins associated with the corresponding subnet, port and severity. The OpenVAS reports provide an overview of affected targets and a brief description of the vulnerability insight, indicate whether the impact level is application- or system-specific, present possible fixes to mitigate vulnerability and its impact on the application or system and provide references related to the detected vulnerabilities. Each threat is associated with vulnerabilities in an OpenVAS notation. The identified vulnerabilities were labeled as high, medium, low, log and false positive based on the Common Vulnerability Scoring System (CVSS) (Mell *et al*., 2007). Table 4 presents the risk factors and their corresponding CVSS base score range. These risk factors are vendor specific;

Table 4: Risk factors based on the CVSS base score

Risk factors	CVSS v2 base scores
High	10 - 7
Medium	6.9 - 4
Low	3.9 - 0.1
Info	0

thus, any severity labeled High in OpenVAS may not have the same level of severity when other scanners are used. Risk factors should hence be regarded merely as guidelines or suggestions because they only reflect the CVSS base score.

RATS setup: RATS can be downloaded from the URL, "code.google.com". It can be downloaded as a WinRAR file for subsequent extraction and run against the source code using Command Prompt. Upon installation, RATS can be utilized to edit software. RATS accepts several command line options (RATS, 2013) and a list of files to audit on the command line. Plain C/C++/Perl/PHP/Python source code can be inputted to RATS. The files must be named appropriately (i.e., .c, .cc, .pl, .php and .py) to ensure that the correct rule set (e.g., the Perl rule set) is employed. The tool's language option must be set to enforce the usage of a specified rule set.

RATS is applicable to both open and closed source software; it helps generate more secure codes by identifying common security issues in the source code and suggesting that more secure codes should be used instead (Cowan, 2003; Nazario, 2002). RATS is configurable and allows for different levels of output (depending on the priority of the issue found) and ignorable lines. RATS identifies more errors compared with other auditing tools. When initialized, RATS scans each file specified on the command line and produces a report when scanning is complete. The vulnerabilities in the final report depend on the data contained in the vulnerability database or databases used and the warning level. The list of files and line numbers where each vulnerability occurred is provided together with a brief description of the vulnerability and the recommended action. RATS prints potential problems sorted by severity, then by function name, then by file and then by line number. RATS prints an explanation of the problem (if available in the vulnerability database) for each function name. Finally, RATS prints the number of lines analyzed and the time spent.

RATS is one of the most versatile tools available because of the number of languages it can scan for vulnerabilities. RATS has the ability to scan many different types of languages and has elegant and detailed reporting features. In summary, RATS is a useful scanner to have in a programmer's toolbox, especially for those who constantly use many different languages (SANS Institute, 2002).

ETHICS AND RULES

Vulnerability assessment was performed on 210 randomly selected web application systems. An ethical penetration requires permission from the owners of the web applications. As the scope of the vulnerability assessment is 210 web applications, difficulties were encountered to attain permission and agreement from the 210 web application owners. In this study, an agreement was thus achieved with one web hosting company to conduct a vulnerability assessment on their web application system. The web hosting company has several services, such as the design, development, maintenance and delivery of websites which meet the business needs of customers aside from providing web domains.

When the agreement was achieved from the web hosting company, rules of engagement had to be observed to present professional, ethical and authorized reports. The following rules define the assessment approach and scope:

- An assessment can be performed beyond the scope. However, voyaging beyond identified boundaries without permissions from the company is prohibited
- The assessment plan concerns the amount of time required to assess the security of a target system. A schedule is established to ensure that assessment does not interrupt business operations
- The assessment processes define the set of steps that must be strictly followed during the assessment. These rules restrict the assessment process for its environment and company staff by combining technical and managerial views
- The assessment results should be presented in a clear manner. The results should identify all known and unknown vulnerabilities and should be delivered confidentially to the authorized individual

The assessment results are discussed in this study for academic purposes only. The names of the web hosting company and individual web applications will not be mentioned in this study or future studies.

With the agreement, the web hosting company had obtained free assessment of vulnerabilities on their web application. The well-defined report represents the assessment's vulnerability analysis results that are useful for further assessment stages. The vulnerability analysis serves as fundamental information for future vulnerability assessment processes. However in this study, the scope is limited to assessing web application vulnerabilities.

OBSERVATIONS

Two observations were considered in the verification process. First, the availability attribute is defined as, "the software should be operational and ready to provide correct service". The reliability attribute is defined as, "the continuity of correct service", (Kahtan *et al*., 2012). Verification of the availability and reliability attributes was thus based on measuring the correct service of the software system. Apache JMeter was utilized to create workload and measure the correct service. Second, integrity, confidentiality and safety attributes are defined as the absence of vulnerabilities in the software system. The maintainability attribute is defined as the ability to undergo repairs and modifications without introducing errors or vulnerabilities (Kahtan *et al*., 2012). Verification of the integrity, confidentiality, safety and maintainability attributes was hence based on detecting the vulnerabilities of the software system. OpenVAS was employed to scan and identify vulnerabilities. OpenVAS has numerous operations that can be launched to determine the integrity, confidentiality and safety of the software system. RATS was selected to identify the common programming errors of the source code. RATS performs security analysis and provides a list of potential trouble spots to focus on. RATS was used in this study to evaluate the maintainability attributes.

RESULTS AND DISCUSSION

VATs were used to identify vulnerabilities in the current web application systems. A total of 210 web application systems were chosen in random for this assessment. The assessment was performed by the VATs method on six dependability attributes, namely, availability, reliability, confidentiality, integrity, safety and maintainability. VATs were thus used for the web application assessment. The assessment tools include Apache JMeter, OpenVAS and RATS. The filtered results of the 210 scans are presented as follows:

- **Apache JMeter results:** JMeter tool was used to measure the availability and reliability attributes of the 210 selected web application systems. After performing the run of 210 test plans, results were gathered and analyzed. Fig. 1 shows the average percentage of 210 web applications for availability and reliability versus DoS. Horizontal axis represents the vulnerabilities per second, whereas the vertical axis represents the average percentage of the 210 accumulated readings. According to Fig. 1, when the number of vulnerabilities (based on user) increases, the availability and reliability percentage of web application systems decrease, whereas, DoS increases gradually. For example, when the vulnerabilities per second is 1, the average percentage of availability and reliability of the system is high because of the lower number of users in the load test. The average percentage of DoS is hence low. In contrast, when the vulnerabilities per second reach 100 and above, the average percentage availability and reliability of a system is low because of the high number of users in the load test. The average percentage of DoS is hence high. This condition implies that the DoS exceeds the capacity because of the inability of a web application system to handle extra requests

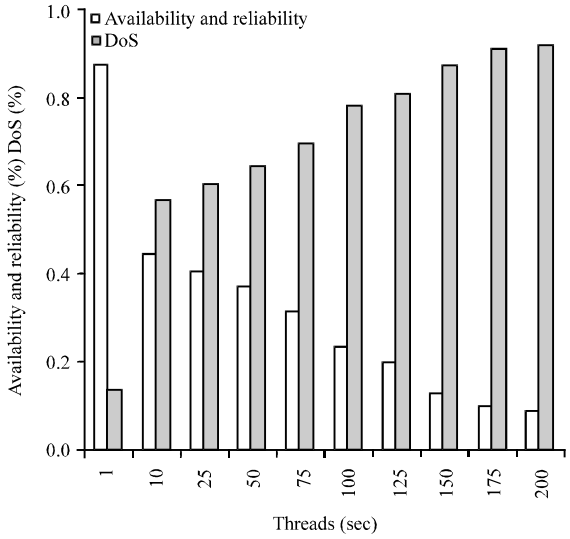

Fig. 1: Average percentage of 210 web application for availability and reliability vs. DoS

- **OpenVAS results:** OpenVAS was used to perform the scan against the 210 web application systems to measure the confidentiality, integrity and safety attributes. From the scans, reports were generated that listed the vulnerabilities detected in the 210 web application systems. Figure 2-4 show the filtered results of the 210 scans. Vulnerabilities marked as Log and False Positive are not included in the figures. The vulnerabilities of confidentiality attributes are presented in Fig. 2. The highest total vulnerabilities detected by OpenVAS are under the Low risk factor, where 74 websites are accounted

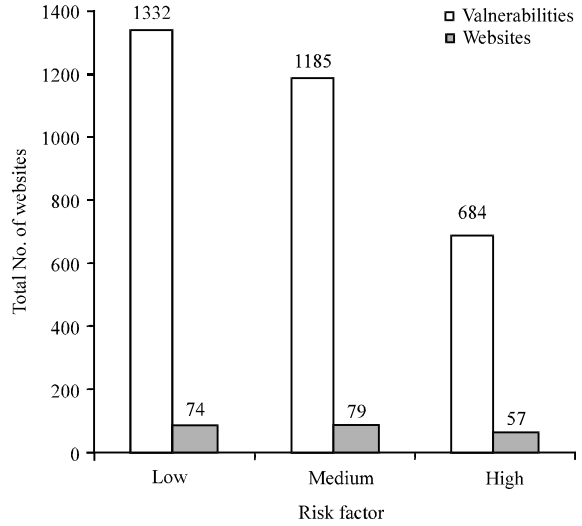

Fig. 2: Confidentiality vulnerabilities

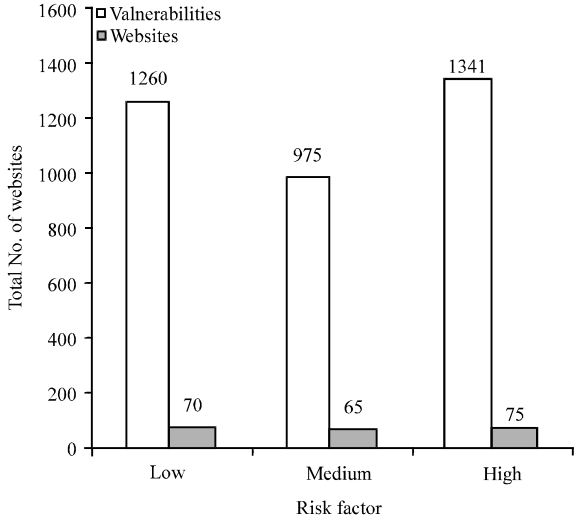

Fig. 3: Integrity vulnerabilities

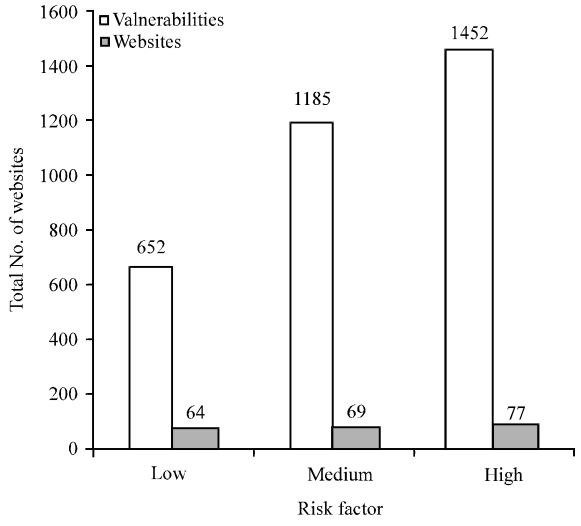

Fig. 4: Safety vulnerabilities

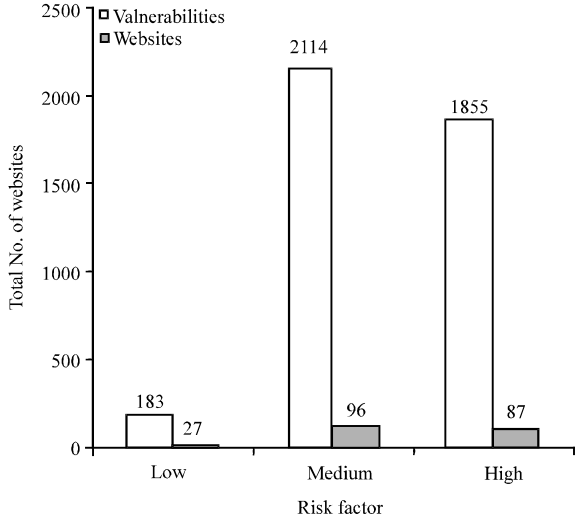

Fig. 5: Maintainability vulnerabilities

for. On the other hand, 79 websites contribute to 1185 total vulnerabilities for the Medium risk factor. As for the lowest total vulnerabilities detected, the high risk factor represents 684 total vulnerabilities for 57 websites.

The vulnerabilities of integrity attributes are also presented in Fig. 3. According to Fig. 3, the highest total vulnerabilities detected by OpenVAS are under the high risk factor, where 75 websites are accounted for. As for the lowest total vulnerabilities detected, the medium risk factor represents 957 total vulnerabilities for 65 websites. Approximately 70 websites contribute to 1260 total vulnerabilities for the Low risk factor.

Figure 4 presents the vulnerabilities of safety attributes. The highest total vulnerabilities detected by OpenVAS are under the High risk factor, where 77 websites are accounted for. As for the lowest total vulnerabilities detected, the Low risk factor represents 652 total vulnerabilities for 64 websites. Approximately 69 websites contribute to 1185 total vulnerabilities for the medium risk factor.

- **RATS results:** RATS was used to perform the scan against the 210 web application systems to measure the maintainability attribute (Fig. 5). From the scans, reports were generated that listed the vulnerabilities

detected in the 210 web application systems. The vulnerabilities of the maintainability attribute are presented in Fig. 5. The highest total vulnerabilities detected by RATS are under the Medium risk factor, where 96 websites are accounted for. As for the lowest total vulnerabilities detected, the low risk factor represents 183 total vulnerabilities for 27 websites. Approximately 87 websites contribute to 1855 total vulnerabilities for the high risk factor

CONCLUSION

A vulnerability assessment was performed on 210 selected web application systems using VATs. In this study, three assessment tools were utilized: Apache JMeter, OpenVAS and RATS. The selected tools are capable of measuring the vulnerabilities that affect the dependability attributes. The vulnerability assessment results indicate that the existing web application systems encounter a high record of vulnerabilities. These vulnerabilities target the dependability attributes. Hence, this scenario will cause system functionality failure, suspension or denial of service that leads to poor system performance or system crash. Dependability attributes must hence be verified and validated throughout the software development process to guarantee the dependability of web applications. The results of the vulnerability assessment are useful in uncovering the weaknesses of the web application system that have been constructed without embedding the dependability attributes into the development process. These results will also increase the awareness of the software development industries on the effects of neglecting dependability attributes in the software development process. Finally, building a solid security strategy to guard against identified software vulnerabilities is easier based on the identified vulnerabilities.

REFERENCES

Alberts, C.J., J.H. Allen and R.W. Stoddard, 2012. Risk-based measurement and analysis: Application to software security. Software Engineering Institute, Carnegie Mellon University, USA., http://repository.cmu.edu/sei/707/

Apache JMeter, 2010. Apache software foundation. http://jmeter.apache.org/usermanual/glossary.html.

Brokken, F.B., 2013. Open source in mixed environments-using open source to improve security and throughput. Proceedings of the Workshop on Creating Awareness for the Use of Open Source Systems in the Public Sector in Afghanistan, September 15-17, 2012, Kabul, Afghanistan.

Cinque, M., D. Cotroneo and A. Pecchia, 2010. Enabling effective dependability evaluation of complex systems via a rule-based logging framework. Int. J. Adv. Software, 2: 323-336.

Colombo, R.T., M.S. Pessoa, A.C. Guerra, A.B. Filho and C.C. Gomes, 2012. Prioritization of software security intangible attributes. ACM SIGSOFT Software Eng. Notes, 37: 1-7.

Cowan, C., 2003. Software security for open-source systems. IEEE Secur. Privacy, 1: 38-45.

Curphey, M. and R. Arawo, 2006. Web application security assessment tools. IEEE Secur. Privacy, 4: 32-41.

EC-Council, 2010. Penetration Testing: Procedures and Methodologies. Vol. 2, Cengage Learning, USA., ISBN-13: 9781435483675, Pages: 240.

Espadas, J., A. Molina, G. Jimenez, M. Molina, R. Ramirez and D. Concha, 2013. A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures. Future Gener. Comput. Syst., 29: 273-286.

Gao, J., K. Manjula, P. Roopa, E. Sumalatha, X. Bai, W.T. Tsai and T. Uehara, 2012. A cloud-based TaaS infrastructure with tools for SaaS validation, performance and scalability evaluation. Proceedings of the 4th International Conference on Cloud Computing Technology and Science, December 3-6, 2012, Taipei, Taiwan, pp: 464-471.

Han, J. and K.M. Khan, 2006. Assessing security properties of software components: A software engineer's perspective. Proceedings of the Australian Software Engineering Conference, April 18-21, 2006, Sydney, Australia, pp: 199-210.

Hartel, P.H., 2012. Review of the Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy. P. Engebretson, Syngress Publishing, Waltham, MA., USA.

Kahtan, H., N.A. Bakar and R. Nordin, 2012. Reviewing the challenges of security features in component based software development models. Proceedings of the IEEE Symposium on E-Learning, E-Management and E-Services, October 21-24, 2012, Kuala Lumpur, Malaysia, pp: 1-6.

Kahtan, H., N.A. Bakar and R. Nordin, 2014a. Awareness of embedding security features into component-based software development model: A survey. J. Comput. Sci.,

Kahtan, H., N.A. Bakar and R. Nordin, 2014b. Dependability attributes for increased security in component-based software development. J. Comput. Sci.

Karen, G., T. Winograd, H.L. McKinley, P. Holley and B.A. Hamilton, 2006. Security in the software life cycle: Making software development processes-and the software produced by them-more secure, draft version 1.1. Department of Homeland Security, July 2006.

Kim, H., 2004. A Framework for Security Assurance in Component Based Development. In: Proceedings of the Computational Science and its Applications, Lagana, A., M.L. Gavrilova, V. Kumar, Y. Mun, C.J.K. Tan and O. Gervasi (Eds.). Springer, Berlin, Germany, pp: 587-596.

Lai, S.T., 2012. An analyzer-based security measurement model for increasing software security. Int. J. Comput. Sci., 4: 81-91.

Manoj, S., V. Kumar and R. Gupta, 2002. Security threat perception to a digital library. http://ir.inflibnet.ac.in/handle/1944/39

McGraw, G., 2004. Software security. IEEE Security Privacy, 2: 80-83.

McGraw, G., 2011. Software security and the building security in maturity model. http://cds.cern.ch/record/1337251

Mell, P., K. Scarfone and S. Romanosky, 2007. CVSS: A complete guide to the common vulnerability scoring system, version 2.0. http://www.first.org/cvss/cvss-guide.pdf

Mir, I.A. and S.M.K. Quadri, 2012. Analysis and evaluating security of component-based software development: A security metrics framework. Int. J. Comput. Network Inform. Secur., 4: 21-31.

Morris, T., S. Pan, J. Lewis, J. Moorhead and B. Reaves *et al.*, 2011. Cybersecurity testing of substation phasor measurement units and phasor data concentrators. Proceecings of the 7th Annual ACM Cyber Security and Information Intelligence Research Workshop, October 12-14, 2011, Oak Ridge, TN., USA.

Nazario, J., 2002. Source code scanners for better code. Linux J.,

OpenVAS, 2013. The open vulnerability assessment system. http://www.openvas.org/about.html

Planquart, J.P., 2001. Application of Neural Networks to Intrusion Detection. SANS Institute, USA.

RATS, 2013. Rough auditing tool for security. https://code.google.com/p/rough-auditing-tool-for-security/.

SANS Institute, 2002. Secure software development and code analysis tools. SANS Institute InfoSec Reading Room, September 30, 2002. http://www.sans.org/reading-room/whitepapers/securecode/secure-software-development-code-analysis-tools-389

Secunia, 2012. Secunia yearly report 2011. http://secunia.com/?action=fetch&filename=Secunia_Yearly_Report_2011.pdf.

Simpson, S., 2012. Sharing lessons learned: Practiced practices for software security. Datenschutz Datensicherheit-DuD, 36: 641-644.

Singh, N., 2012. A component-based model for E-business, integrating knowledge management and E-commerce. J. Inform. Oper. Manage., 3: 25-28.

Sophos, 2012. Security threat report 2012. http://www.sophos.com/medialibrary/PDFs/other/SophosSecurityThreatReport2012.pdf.

Steward, C., L.A. Wahsheh, A. Ahmad, J.M. Graham, C.V. Hinds, A.T. Williams and S.J. DeLoatch, 2012. Software security: The dangerous afterthought. Proceedings of the 9th International Conference on Information Technology: New Generations, April 16-18, 2012, Las Vegas, NV., USA., pp: 815-818.

Tevis, J.E.J. and J.A. Hamilton, 2004. Methods for the prevention, detection and removal of software security vulnerabilities. Proceedings of the 42nd Annual Southeast Regional Conference, April 2-3, 2004, Huntsville, AL., USA., pp: 197-202.

Umrao, S., M. Kaur and G.K. Gupta, 2012. Vulnerability assessment and penetration testing. Int. J. Comput. Commun. Technol., 3: 71-74.

Verizon, 2012. Data breach investigations report. http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigations-report-2012-ebk_en_xg.pdf

Wales, E., 2003. Vulnerability assessment tools. Network Secur., 2003: 15-17.