A Semi Automated Approach to Assess Web Vulnerability Scanner Tools Effectiveness

Tania Basso[1], Regina L. O. Moraes[2], Mario Jino[1]

[1]Faculdade de Engenharia Elétrica e de Computação
[2]Faculdade de Tecnologia
Universidade Estadual de Campinas (UNICAMP) – Campinas, SP, Brazil
{taniabasso, regina}@ft.unicamp.br, jino@dca.fee.unicamp.br

Abstract. *Nowadays, software products are developed with security vulnerabilities due to bad coding. Vulnerability scanner tools automatically detect security vulnerabilities in web applications; thus, trustworthiness on the results of these tools is essential and, sometimes, the evaluation of their results is done manually or even empirically. This work presents a semi automated approach, based on fault injection techniques, to assess the efficacy of these tools. Three scanner tools were assessed with the presence of realistic software faults responsible for security vulnerabilities in web applications. Results show that the approach is effective and has the advantage of predicting security vulnerabilities through the fault injection techniques.*

1. Introduction

The World Wide Web has become a sophisticated platform that is capable of delivering a broad range of web applications. From single individuals up to large organizations, there is an increasing dependency on this technology. Information and data are stored, traded and made available on the Web. This type of application is becoming increasingly exposed, the reason why as any security vulnerability can be exploited by crackers. The consequences can range from simple website defacement to environmental disasters and loss of human life [Gilman 2009]. Also, attacks to organizations can put in check their credibility and have a highly negative impact on users. Therefore, security and reliability have become a priority for web applications.

Software faults are mistakes made by software product programmers and remain in the program source code. The root cause of most security attacks are security vulnerabilities created by software faults [Fonseca and Vieira 2008, Basso et al. 2009]. Even though the software developers are encouraged to follow best coding practices – and it includes security aspects – most times, due to time restrictions, the developers are focused on developing the main functionalities and satisfying the client requirements, neglecting security aspects and introducing software faults that can be responsible for security vulnerabilities. It is known that traditional network security mechanisms as firewalls, cryptography and intrusion detection systems can protect the network but not mitigate web application attacks. Then, the attackers are changing their focus from the

 ©2013 SBC — Soc. Bras. de Computação

network to the web applications, where the insecure codification represents major risks [Fonseca and Vieira 2008].

Automatic vulnerability scanners are tools often used by developers and system administrators to test Web applications against security vulnerabilities. Reliable results from vulnerability scanners are essential and the analysis of the scanners' effectiveness is important to guide the selection as well as the use of these tools. Also, scanner tools are developed by organizations [Acunetix 2013, HP Webinspect 2013, IBM Security AppScan 2013] and academic researchers [Chen and Wu 2010, Galán et al. 2010] and must be tested to evaluate its effectiveness, especially when is desired to compare their results with results provided by other scanner tools (and demonstrate which has the best ones). Sometimes, the tests are done manually [Chen and Wu 2010], with targeted web applications and usually they do not consider predicting software faults which lead to security vulnerability nor use a flexible approach to include new ways of attack that arises [Chen and Wu 2010, Galán et al. 2010].

The goal of this paper is to show a semi automated approach to assess the efficacy of vulnerability scanner tools. This approach is based on software fault injection and attack trees modeling. It consists of injecting realistic Java software faults into web applications and, once the faults are injected, the scan is run to check if it can detect the potential vulnerabilities caused by the injected fault. Existence of vulnerabilities is confirmed through attacks, guided by the attack trees, which can give the attacker vision to exploit the security vulnerabilities in the web application. The procedures of fault injection and attack injection are automated and performed through the J-SWIFT [Sanches et al. 2011] and J-Attack [Fernandes et al. 2011] tools. The Java programming language was chosen due to its extensive use for developing modern applications, especially web applications.

The structure of the paper is as follows: Section 2 presents the related work, which established the basis of this work; Section 3 describes the tools and technologies used to compose the semi automated approach. Section 4 presents the approach and section 5 presents an experimental study to demonstrate it, regarding the absence and the presence of injected faults and the scalability of using attack trees. Section 6 presents the discussion about the results of the case study and Section 7 presents conclusions and future works.

2. Related work

Some works bring new developed vulnerability scanner tool and some works assess the effectiveness of available scanner tools. In both cases is necessary to have a way of evaluating the effectiveness of the results. Effectiveness may be assessed by two main aspects: vulnerability coverage and false positive rate. Vulnerability coverage refers to the ability of the tool to detect correctly all security vulnerabilities in the application (the scanner is considered doubtful whether undetected vulnerabilities do not really exist in the application or the scanner was not able to detect it). False positive refers to vulnerabilities detected by the scanner tool that, in fact, do not exist in the application.

Chen and Wu [Chen and Wu 2010] developed an automated vulnerability scanner system based on injection point which automatically analyses web sites with the aim of finding exploitable SQL injection and XSS vulnerabilities. To evaluate the

 ©2013 SBC — Soc. Bras. de Computação

reliability of their system, they designed a simple targeted web application with many SQL-injection and XSS vulnerabilities and use other seven applications. Although the first application can be used in a controlled way, the other ones can have more unknown vulnerabilities, which can affect the results of the developed tool.

Bau et al. [Bau et al. 2010] developed a study to assess the state of the art of the vulnerability scanners. The authors evaluated the efficacy of eight commercial scanners vulnerability tools on detecting security vulnerabilities like Cross Site Scripting (XSS), SQL Injection and Cross Site Request Forgery (CSRF). The validation of the results was based on already known vulnerabilities in applications under test. The knowledge of these vulnerabilities is from patches developed to correct them. It means that, if a particular release of an application has a correction patch for a particular vulnerability, it is because the application does have this vulnerability and the scanner should detect it. However, the authors do not predict vulnerabilities caused by hidden software faults, which can provide more accurate results. Fonseca et al. [Fonseca et al. 2007] also present an experimental evaluation of security vulnerabilities. Four well known vulnerability scanners have been used to identify security flaws in web services implementations. They used the fault injection technique and the existence (or not) of each vulnerability detected was confirmed manually. Also, this study was focused on a specific family of applications, namely database centric web based applications written in PHP, and the results obtained cannot be easily generalized, especially if we take into account the specificities of web services environments.

Fonseca et al. [Fonseca et al. 2009] proposed a methodology to automatically inject vulnerabilities and attacks in web applications to assess security mechanisms in place. Two commercial web application vulnerability scanners were assessed and, however the methodology is automated, it does not permit the scalability of the attacks, i.e., when new forms of attacks arise, is not easy to insert them in the attack tool. Nor other different attack types. Our intention is to propose a clear and semi automated approach where, following some stages, it is possible to evaluate the effectiveness of any scanner vulnerability tool, including the prediction of vulnerabilities caused by hidden software faults (through software fault injection techniques) and scalability of attacks (through attack trees models). This scalability is achieved through an attack injection tool based on attack trees, which permit the inclusion of new ways of performing particular attacks as they arise. So, it is possible to have a more complete attack injection tool and better results in assessing the vulnerability scanner tools.

As results, the previous works [Bau et al. 2006, Fonseca et al. 2007, Fonseca et al. 2009] show, by agreement, that the scanners had low coverage and several cases of false positives, indicating the limitations of this segment of tools.

3. Technologies and Automatic Tools

The proposed approach can be used with different tools. The two automatic tools were used are the J-SWFIT [Sanches et al. 2011] and J-Attack [Fernandes et al. 2011]. They are based, respectively, on technologies as software fault injection and attack trees, which are briefly described below.

 ©2013 SBC — Soc. Bras. de Computação

3.1. Fault Injection Techniques and the J-SWFIT tool

Software faults are mistakes made by software product programmers and remain in the program source code. The complete elimination of software faults is a difficult or even impossible goal to be achieved [Lyu 1996, Musa 1996]. Fault injection techniques are one option for activation of these faults in order to evaluate the software behavior during the software validation process. This technique consists of introducing faults into a target system in a deliberated way and observing if the system keeps operating as desired [Hsueh et al. 1997]. In some cases, software faults are responsible for security vulnerabilities that can be exploited. The use of fault injection to assess security is actually a particular case of software fault injection. It helps to speed up the detection of security vulnerabilities, allowing that countermeasures are applied to eliminate them or to reduce the severity of their exploitation, contributing to higher levels of dependability for the application under test.

Knowledge of the software fault representativeness is essential for realistic fault injection. A big concern when using fault injection environments and tools is to ensure that injected faults represent real faults, because it is necessary to obtain significant results. Thus, defining a realist faultload (i.e., a set of selected faults to be injected) is primordial to the success of the dependability validation of software systems. Basso et al. [Basso et al. 2009] presented a representative faultload to security software Java faults, with faults that represents specificities of the language. The most frequent faults of the security fautload were implemented in the J-SWFIT tool.

The Java Software Fault Injection Tool – J-SWFIT allows the automatic injection of software faults in Java systems in a scalable way. This tool is based on a set of fault injection operators (implemented from the faultload) that reproduce directly in the target bytecode the instruction sequences that represent the most common types of high-level software faults. So, the J-SWFIT preserves the independence of the availability of the source code to inject software faults. The operation of J-SWFIT consists of: finding places where specific software faults can exist; injecting each software fault independently; executing the application with the present software fault and monitoring the results; at the end, the tool compares the behavior of the application on the presence and absence of each fault.

3.2. Attack trees and the J-Attack tool

Attack tree [Schneier 1999] is a structure that permits to represent, in a clear and organized way, the attacker actions to exploit security vulnerability successfully. In the attack trees the root node represents the achievement of the ultimate goal of the attack. Each child node represents sub-goals that have to be accomplished in order to reach the parent's goal to succeed. Parent nodes can establish relationships with their child nodes using an "OR" or an "AND" relationship. In an "OR" relationship, if any of the child nodes sub-goals are accomplished then the parent node is successful. With an "AND" relationship, all of the child node sub-goals must be accomplished in order to the parent node to be successful. The leaves of the tree (i.e., nodes that are no longer decomposed) represent attacker's actions.

J-Attack is a tool developed to automate injection attacks through the model tree. In J-Attack, the test cases extracted from attack trees are registered in a database that is

 ©2013 SBC — Soc. Bras. de Computação

used to guide the tool to perform the tests. To inject attacks, the user have to inform the target application and then the application is mapped, i.e., the tool sends request to the URL provided by the user, identifies and stores links contained in the reply to the initial request. For each link, the input fields are identified and the tests registered in the database are executed. J-Attack is scalable, so, it is possible to expand it to cover more security vulnerabilities. New tests and vulnerabilities can be added to the database.

4. The Proposed Approach

This section presents the proposed approach and its respective stages for evaluation of the effectiveness of vulnerability scanner tools by the two main aspects (vulnerability coverage and false positive rate). The goal is to provide guidelines for the acquisition and use of these tools, and, thereby, provide higher levels of dependability for Web applications (in this case, web applications developed in Java programming language).

Comparing to other existing techniques, in summary, the approach proposed in this paper presents several key advantages, including:

- It **predicts vulnerabilities caused by hidden software faults** (through the use of software fault injection techniques).
- It **does not require the source code of the web application**, once that the J-SWFIT injects software faults in the compiled source code.
- It is **faster and more accurate** than manual approaches, as the process of injecting faults and attacks are automated through the respective tools.
- It is **extendable to other types of attacks** injection by simply registering it in the J-Attack's database.
- It **does not rely on any learning process**, thus it does not suffer from incomplete or incorrect learning processes, which many times lead to inaccurate results.
- It achieves extremely good results at a **very low financial cost**. Indeed the cost of the tools is null as it is available for free use at [Moraes 2013]. It is necessary only an expert user to apply the approach.

The approach has six stages, illustrated on Figure 1 and described below.

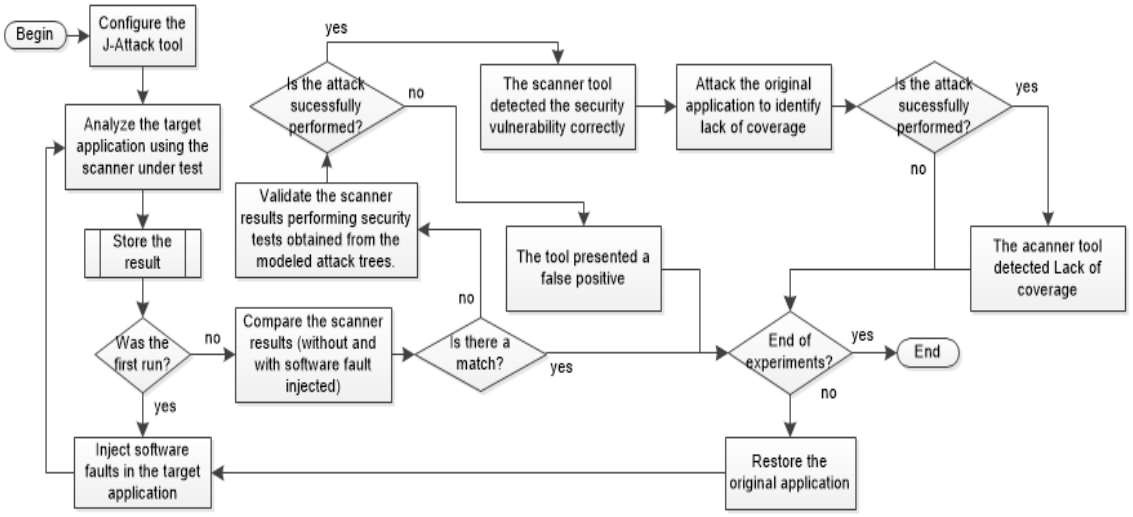

Figure 1. The stages of the semi automated approach.

 ©2013 SBC — Soc. Bras. de Computação

The stage 1 is transparent to the user unless it is necessary to test vulnerabilities that are not configured in J-Attack tool. The stage is described as part of the approach because they can be followed if it is necessary to update the J-Attack with new ways that may arise of exploiting the vulnerability under test or to add different types of attack to exploit different vulnerabilities.

1 – **Configure the J-Attack tool**. If necessary, the J-Attack can be easily complemented with new ways of exploiting vulnerabilities that may arise**.** For this purpose, it is necessary, first, to identify and understand the ways the application can be attacked. Then, the attack trees structure is used to organize and describe the attacks identified in the first step. Different attack paths can be represented in the same attack tree and new attacks can be added due to its modular structure, permitting its expansion. The attack scenarios are derived from the attack trees and registered in the J-Attack's database to perform the attacks automatically. The more complete is the range of attacks to a particular vulnerability, the more efficient is the application of the approach.

2 – **Analyze the target application using the scanner under test and store the results**. In this stage the original web application should be used first, i.e., the web application without any fault injected. Then, with the scanner tool to be evaluated, a scan is performed in this web application – this performing is called "gold run" – and the obtained results should be stored to be used as a basis to compare the next scan results (in the presence of injected software faults). Originally, the web application may have some security vulnerabilities and this scan should detect them, ensuring that new vulnerabilities detected over the experiments are due to the injected faults. This first scan may be the reference to verify the effect of the new software faults injected, especially in the creation of new security vulnerabilities.

3 – **Inject software faults in the target application**. From a defined faultload, the software faults should be injected in the web application. This stage is important because it permits a control about the software faults present in the applications' source code, knowing where the faults are located. The representativeness of the faultload is primordial to the success of assuring new vulnerabilities, i. e., the software faults to be injected must be responsible for creating potential security vulnerabilities.

If new security vulnerabilities are identified due to the fault injected, impacting the application behavior, it is possible to verify the effectiveness of the scanner tool on detecting the new vulnerability. The faults are automatically injected by the J-SWFIT tool.

4 – **Reanalyze the target application (with the software fault injected) using the same scanner**. In this stage the scan may be re-executed in the application in the presence of the software fault injected. Many executions are done, one execution to each injected fault. The obtained results should be stored similarly as described in the stage 2, facilitating the automation of a comparison of results obtained in this stage with the results of the "gold run".

5 – **Compare the scanner results from the original target application (without software fault injected) and the scanner results from the target application with software fault injected**. The results of the original scan (gold run – step 3) may be compared with the results of each scan executed in the web application

 ©2013 SBC — Soc. Bras. de Computação

with software fault injected. This comparison is done to evaluate if there are changes and, if changes are observed, the differences may be investigated because it can indicate false positive, lack of coverage from the scanner tool or new security vulnerabilities due to the presence of software faults. It is important that the scan results from the original web application also be analyzed. The analysis is described in the next stage.

6 – **Validate the scanner results performing security tests obtained from the modeled attack trees**. If there are differences between the original scan results and the scan results in the application with the faults injected, attacks may be performed to the security vulnerabilities that are desired to investigate. These attacks are performed automatically by the J-Attack tool. If the attack is successfully performed it means that the scanner tool detected the security vulnerability correctly. In this case, the attack may be performed also in the original web application to verify if the vulnerability already existed before the fault injected and was not prior detect by the scanner tool, indicating lack of coverage from the tool. Moreover, if the attack is not successfully performed, the tool presented a false positive.

A case study was developed to assess the applicability of the approach and is related in the next section.

5. Case Study

To develop the case study experiments some decisions were taken: the Java web applications were selected; the security vulnerabilities, to be investigated, were identified and configured in the J-Attack tool; the scanner tools, to be evaluated, and the faultload were identified and applied to the J-SWFIT tool.

Three Web applications developed in Java were selected to carry out the experiments. The first one has 137 classes and approximately 27,048 lines of code (LOC), the second one has 262 classes and approximately 45,800 lines of code and the third one has 164 classes, approximately 45 thousands lines of code and four databases integrated.

The first web application is a Customer Relationship Manager (CRM) and Project Management Tool. It uses MySQL database and technologies such as Hibernate, framework Struts and Jasper Reports. The second Web application is a Distance Education management system, developed by the Brazilian federal government. It uses the Postgres database, Hibernate and Ajax technologies. The third web application is from a large company that is focused in human health care. It uses Hibernate, Ajax and the framework Java Server Faces. The database used is MySQL. All web applications were selected because they represent the same segment of applications (data management), and they use some technologies that probably impact the scanner tool results (for example, the Hibernate, which deal with query construction and can affect the detection of SQL injection vulnerabilities). Moreover, they are web applications whose services are applied in the commercial and academic areas. We have chosen similar use cases from all applications to be the target piece of code of injected faults, as create, retrieve, update and delete products, courses and appointments, respectively.

The three security vulnerabilities considered for this study are SQL injection, XSS (Cross Site Scripting) and CSRF (Cross Site Request Forgery). They were selected

 ©2013 SBC — Soc. Bras. de Computação

because of their criticality, occupying the first, second and fifth place in the 2010 OWASP Top 10 [OWASP 2010] and first, third and eighth places in the 2013 OWASP TOP 10 [OWASP 2013] respectively. These vulnerabilities are widely spread and dangerous, and may cause major damage to the victims. Although the CSRF vulnerability type represents the fifth place in 2010 and eighth place in 2013 more critical top 10 rank, testing Web applications against this type of vulnerability is important because the attacks are difficult to detect and, usually, the remediation is only possible after the incidents. Moreover, according to Lin et al. [Lin et al. 2009], this is an area in which limited research has been done.

XSS occurs when a web application gathers malicious data from a user, usually gathered in the form of a hyperlink which contains malicious content within it. After the data is collected by the Web application, it creates an output page for the user, containing the malicious data that was originally sent to it, but in a manner to make it appear as valid content from the website [Uto and Melo 2009] . SQL injection refers to a class of code-injection attacks in which data provided by the user is included in an SQL query in such a way that part of the user's input is treated as SQL code. By leveraging these vulnerabilities, an attacker can submit SQL commands directly to the database [Halfond et al. 2006]. Last, CSRF works by exploiting the trust a site has for the user. Site tasks are usually linked to specific URLs allowing specific actions to be performed when requested. If a user is logged into the site and an attacker tricks their browser into making a request to one of these task URLs, then the task is performed and logged as the logged in user [CSRF 2013].

For each of these three types of vulnerability an attack tree was created (see Fernandes et al. [Fernandes et al. 2010]) and the attack scenarios were registered on the database of J-Attack. For the CSRF tree we covered the part of the CSRF attack relative to the acceptance of the requests coming from another source. The part relative to the means used to lure the user to activate the request is not covered as they are out of the defensive bounds that an application can have against CSRF.

The types of fault to be injected are the two most frequent ones from the faultload of Basso et al. [Basso et al. 2009] and are implemented by the J-SWFIT tool. The scanner tools selected were Acunetix [Acunetix 2013] version 6.0, Rational AppScan [IBM Security AppScan 2013] version 7.8 e HP WebInspect [HP Webinspect 2013] version 8.0. These tools were selected due to its great market insertion.

6. Results and Discussions

To present the results of the experiments, the brands and versions of the scanner are not associated, as well as the web applications, to assure neutrality and also because commercial tools providers usually do not permit the publication of the results of this type of evaluation. Then, the scanners used in the experiments will be referred, from this point, as S1, S2 and S3, without any special order. The applications will be called Ap1, Ap2 and Ap3 also without any special order. Results are presented and discussed by means of the effect of fault injection have on the vulnerability results and the lack of coverage and false positives pointed by the scanner tools.

Effect of fault injection on the results. For the three Web applications, we analyzed, respectively, 31, 35 and 24 different scenarios. Each scenario is represented by

 ©2013 SBC — Soc. Bras. de Computação

one fault injected in the selected classes from the use case . The second line from Table 1 shows the total of scenarios that presented new security vulnerabilities due to the fault injection.

Table 1. Applications scenarios and vulnerabilities

	Ap1	Ap2	Ap3	Total
Total scenarios analyzed	31	35	24	90
Scenarios with new vulnerabilities	22	20	0	42
% of faults that affected the scan	70.9%	57.1%	0%	46.7%

According to Table 1, about 46.7% of the injected software faults affected the scanner results. In approximately 20% of the 90 scenarios, the changes in the results were verified by at least two scanners. The injected faults affected the applications behavior and, consequently, the scanner tool behavior, due to the context of the application and the procedures necessary to activate the fault. For example, many faults were injected in locations where a null entry point is verified in the source code. Activating this fault, the application modifies its behavior by not verifying the null entry point and forcing the application to display error pages.

Lack of coverage and false positives. Table 2 shows, for each scanner and each web application, the lack of coverage and the number of false positives obtained in the experiments. The lack of coverage is about vulnerabilities that do exist in the web applications, confirmed through successful attacks and was accounted based on the total number of vulnerabilities detected correctly by the tests using the J-Attack. The number of false positives is related to vulnerabilities indicated by the tool that were not confirmed by the attacks. These attacks were performed using the J-Attack and, to provide greater reliability of the tests – once it depends of the accuracy of this tool – they were also confirmed manually.

Table 2. Applications lack of coverage and false positives

	Lack of coverage				False positive			
	Ap1	Ap2	Ap3	Tot.	Ap1	Ap2	Ap3	Tot.
S1	19	3	1	23	13	1	2	16
S2	54	17	2	73	1	0	0	1
S3	54	10	2	66	18	1	0	19

Based on Table 2, the Ap1 presented the biggest lack of coverage and false positives no matter the scanner tool that was used. By the analysis of the context of the source code, we believe that the reason for this result is that Ap2 and Ap3 code is more modularized, with less coupling with other modules and fewer implemented use cases. Also, the Ap3 is widely used in the commercial environment, so, it is more mature and with less bugs and vulnerabilities.

Table 3 shows the percentage of lack of coverage and false positives according to each type of security vulnerability. The tests based on J-Attack (and, also, manually, using the attack scenarios, to reinforce the existence of the vulnerability) allowed confirming that do exist, among the 90 scenarios analyzed, 3 XSS vulnerabilities, no SQL injection vulnerability and 73 CSRF vulnerabilities. The false positive and lack of coverage rates are calculated from these numbers.

 ©2013 SBC — Soc. Bras. de Computação

Table 3. Lack of coverage and false positive rate by security vulnerability type

		Correctly detected	Lack of coverage	False positive
S1	XSS	100%	0%	0%
	Inj. SQL	----	0%	100%
	CSRF	58.8%	27.0%	14.2%
S2	XSS	75.0%	0%	25.0%
	Inj. SQL	----	0%	0%
	CSRF	0%	100%	0%
S3	XSS	33.3%	66.7%	0%
	Inj. SQL	----	0%	100%
	CSRF	12.20%	86.5%	1.3%

In Table 3 it is possible to observe that the highest rate of lack of coverage refers to CSRF vulnerabilities and encompasses the three scanners, S1, S2, and S3, with respectively 27.0%, 100% e 86.5%. The lacks of coverage were identified in the original applications (without any fault injected) and in the applications with faults injected. In most of the cases, when scanning the application with faults injected, a new vulnerability detected by the tool was the one that was already presented in the original application, not identified in the "Gold Run". The false positives are about the three types of vulnerabilities, representing 25% to XSS (detected by scanner S2), 100% to SQL injection (detected by scanners S1 and S3) and approximately 16% to CSRF (14.2% detected by scanner S1 and 1.3% detected by S3).

The false positive associated to the XSS vulnerabilities is considered because the scanner tool integrates outdated version of internet browsers. An attack successfully executed by the tool, when executed in the later versions of internet browsers, has no effect, because these versions implement features that do not permit the execution of common XSS attacks.

The SQL injection false positives were identified through the attacks and the analysis of the source code. All applications use the Hibernate technology, which is an object/relational persistence and query service. It permits to encapsulate the queries and send objects to the database through predefined classes and methods, discarding the necessity of explicit SQL queries constructions. The code constructed with Hibernate is more difficult – but not impossible – to have vulnerability to SQL injection attacks. However, the way that the applications were coded, i.e., extremely encapsulated, do not open opportunities to develop successful attacks. Even the scanner tool provides no assurance about its detection result, and it informs that this detected vulnerability requires user verification.

In most of the cases where CSRF false positives were identified, they happened in error pages. An attacker performing a CSRF attack on error pages can be dangerous if the error page presents links or buttons that permit access to the application (as "back" buttons which bring back the user to the last page he/she accessed) or if the error page displays private information about the system (such as database name or table names). For all three applications, the error pages do not present any way of accessing application functionalities or private information. Hence, we considered these cases as false positives because a CSRF attack when accessing the error pages is useless.

 ©2013 SBC — Soc. Bras. de Computação

Also it is possible to observe in Table 3 that CSRF vulnerabilities are more frequent than XSS and SQL injection. This is due to the development frameworks as Struts or Java Server Faces and data persistence technologies as Hibernate that was used in the web applications. These development frameworks have, as default, a filter to execute scripts. So, unless the programmer, when constructing the application code, specify that the application may execute scripts – and it is not common because the idea is to add security aspects to the web application instead of removing them – the application will treat any script as a string, i.e., tags as <script>/<script> will not be executed, but added to the application as a normal text. About Hibernate, its use permits that the application has its queries totally encapsulated in objects and API (Application Programming Interfaces), not allowing that pieces of query be concatenated.

Efficacy of the evaluated scanners. Figure 2 illustrates, for each type of security vulnerability analyzed, the relationship between the detections by the scanners S1, S2 and S3. The intersection areas in the circle represent the number of the same vulnerabilities detected by more than one scanner. The cases of lacks of coverage were not considered to the illustration, i.e., only the detected vulnerabilities were considered, including the false positives.

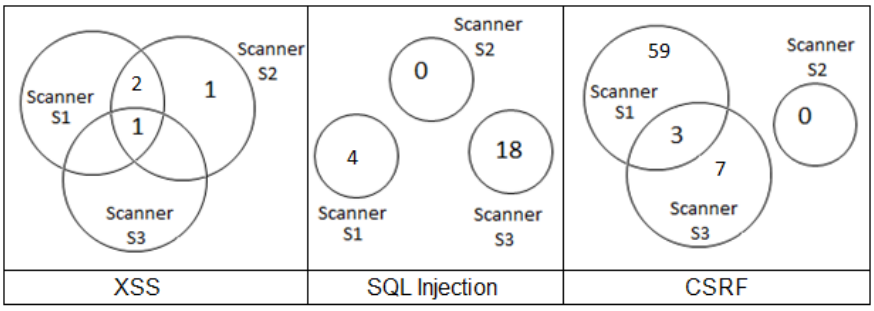

Figure 2. Relationship between the security vulnerability detection

According to Figure 2, there are few vulnerabilities in common between the three scanner tools. This difference indicates that these tools implements different ways of performing the intrusion tests and the results from different tools can be very different. It suggests that, to have a good coverage to security tests, the user can combine multiple scanner tools instead of trusting the results of only one. An analysis of the efficacy of these scanner tools can provide guidelines to their selection. Table 4 shows the percentage of lack of coverage and false positives to each scanner tool analyzed, no matter the type of vulnerability.

Table 4. Lack of coverage and false positives from the scanner tools.

	S1	S2	S3
Vulnerabilities analyzed	92	77	95
Vulnerabilities correctly detected	53 (57.6 %)	3 (3.8 %)	10 (10.5%)
Lack of coverage	23 (25.0 %)	73 (94.9 %)	66 (69.5%)
False positive	16 (17.4 %)	1 (1.3 %)	19 (20.0%)

While comparing the aspects of coverage and false positives that the tools presented, it is clear, by looking at Table 4, the scanner S1 is the best in terms of coverage, presenting the lowest rate in cases of lack of coverage (25.0%), followed by

 ©2013 SBC — Soc. Bras. de Computação

scanners S3 and S2, with, respectively, 69.5% and 94.9%. The scanner S2 presented the highest rate of lack of coverage but, however, presented much lower percentage of false positives compared to scanners S1 and S3 (1.3%). The percentage of false positives from scanners S1 and S3 are close (17.4% and 20.0% respectively), considering the higher number of vulnerabilities that has been analyzed for the scanner S3.

It is also possible, observing Table 4, to evaluate the effectiveness of the scanners according to the type of vulnerability. The scanner S1 presented better results on testing vulnerabilities like XSS because it did not present false positives or lack of coverage for this type of vulnerability. The scanner S2 presented better results for SQL injection vulnerabilities because it also did not detect any false positive or lack of coverage. As for CSRF vulnerabilities, the three scanners presented different results, where the number of cases of lack of coverage and false positives are inversely proportional (the lower the lack of coverage rate, the higher the incidence of false positives and vice versa). In this case, other selections criteria should be used in a complementary way, such as a second most important vulnerability to determine the best tool to be adopted.

As expected, the rate of false positives tends to be directly proportional to the capacity of the tool to detect vulnerabilities. These results, which presented high levels of coverage and lack of false positives, show that the application of the approach is adequate and brought similar results to previous works (cited in section II), reinforcing the evidence of the limitations of scanners vulnerabilities. To critical web applications, multiple scanners should be used and complementary tests may not be discarded.

7. Conclusions

In today's scenario that portrays the lack of security in web applications – especially in the management information segment – and the limitations of vulnerability scanner tools that were presented in this work and the previous ones, that are due to the low effectiveness of the tools, something must be done. The present paper proposes a semi automated approach to validate the effectiveness of these kinds of tools and, with this, to contribute to increase the level of dependability of the web applications. The approach has six stages and it is based on fault injection techniques and attack tree models. Previous studies served as the basis to the faultload to be injected; the attack trees served to guide the tests and automatic tools that support the stages of fault injection and attack injection.

A case study was performed to evaluate the effectiveness of three commercial scanner tools with great market insertion. The results showed that the approach is adequate, providing similar results to previous studies, i.e., the low coverage and high false positive rates pointed by the evaluated scanner tools. The advantage of using the approach is to have support tools to automate some stages and to easy the prediction of existent vulnerabilities through software faults injection. Also, especially, the use of attack trees permit the inclusion of new attacks to be performed by the attack tool, improving greater completeness of the security tests and, consequently, better results.

The application of the approach provide guidelines for the selection of the scanner tools, indicating that, to have good coverage of the security tests, the user can combine multiple scanners instead of trusting the results from only one of them. Or even

 ©2013 SBC — Soc. Bras. de Computação

select the most convenient tool in accordance with predetermined priorities, such as the priority in detecting particular type of vulnerability (some scanners presented better results in detecting particular type of vulnerabilities). Additional tests must not be discarded from the procedure, especially for critical web applications.

Although a few web applications were used in the case study, and these applications represent only a segment among the various existing, the approach showed satisfactory results, meeting the goals it has set itself. As a possible limitation of the approach, the false positive identification is dependent on the accuracy of the J-Attack tool. However, this tool has shown good accuracy in previous tests and it was confirmed through our manual analysis. Generally speaking, the set (approach and automatic tools) presents a good technique to perform security tests, even when the applications' source code is not available.

As future work we intend to apply the approach to other segments of web applications to investigate its behavior and generalize the results. If necessary, some adjustments can be performed.

References

Acunetix Web Security Scanner (2013). Available: http://www.acunetix.com/. Accessed: 07-mar-2013.

Basso, T.; Moraes, R.; Sanches, B.; Jino, M. (2009). "An Investigation of Java Faults Operators Derived from a Field Data Study on Java Software Faults". In: Workshop de Testes e Tolerância a Falhas - WTF, Brazil, pp. 1-13.

Bau, J.; Bursztein, E.; Gupta, D.; Mitchell, J.(2010). "State of the Art: Automated Black-Box Web Application Vulnerability Testing". In. IEEE Symposium on Security and Privacy, Oakland, USA. p. 332-345.

Chen, J.-M.; Wu, C.-L. (2010). "An automated vulnerability scanner for injection attack based on injection point". International Computer Symposium (ICS), p. 113 -118.

CSRF (2013). "The Cross-Site Request Forgery (CSRF/XSRF) FAQ". Available: http://www.cgisecurity.com/csrf-faq.html. Accessed: 07-jun-2013.

Fernandes, P. C. S. ; Basso, T. ; Moraes, R. (2011). "J-Attack - Injetor de Ataques para Avaliação de Segurança de Aplicações Web". XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - Workshop de Testes e Tolerância a Falhas, Campo Grande, Brasil.

Fernandes, P. C.; Basso, T.; Moraes, R.; Jino, M. (2010). "Attack Trees Modeling for Security Tests in Web Applications", 4th. Brazilian Workshop on Systematic and Automated Software Testing (SAST). Natal - RN, Brasil.

Fonseca, J. and Vieira, M. (2008) "Mapping software faults with web security vulnerability". IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), Anchorage, USA, p. 257-266

Fonseca, J.; Vieira, M.; Madeira, H. (2007). "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks", in 13th Pacific Rim International Symposium on Dependable Computing - PRDC, p. 365 -372.

 ©2013 SBC — Soc. Bras. de Computação

Fonseca, J.; Vieira, M.; Madeira, H. (2009). "Vulnerability & attack injection for web applications", in IEEE/IFIP International Conference on Dependable Systems Networks - DSN, p. 93 -102.

Galán, E.; Alcaide, E.A.; Orfila, A.; Blasco, J. (2010). "A multi-agent scanner to detect stored-XSS vulnerabilities". International Conference for Internet Technology and Secured Transactions (ICITST), p. 1 -6.

Gilman, N. (2009). "Hacking goes pro". Engineering Technology, vol. 4, nº 3, p. 26 -29.

Halfond, W. G.; Viegas, J.; Orso, A. (2006) "A Classification of SQL-Injection Attacks and Countermeasures", In Proceedings of the International Symposium on Secure Software Engineering - ISSSE, Arlington, Virginia.

HP WebInspect (2013). Available: http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect. Accessed: 07-jul-2013.

Hsueh, M. C.; Tsai, T. K.; Iyer, R. K. (1997). "Fault injection techniques and tools", Computer, vol. 30, no. 4, p. 75–82.

IBM Security AppScan (2013). Available: http://www-01.ibm.com/software/awdtools/appscan/. Accessed: 07-jul-2013.

Lin, X., Zavarsky, P., Ruhl, R., and Lindskog, D. (2009) "Threat Modeling for CSRF Attacks." Proceedings of the 2009 international Conference on Computational Science and Engineering, pp 486-491.

Lyu, M. R. and others (1996). "Handbook of software reliability engineering", vol. 3. IEEE Computer Society Press CA.

Moraes, R. (2013). Available: http://www.ft.unicamp.br/~regina. Accessed: 09-apr-2013.

Musa, J. D. (1996). "Software reliability-engineered testing", Computer, vol. 29, no. 11, p. 61–68.

OWASP (2010). "The Open Web Application Security Project". TOP 10 2010. Available: https://www.owasp.org/index.php/Top_10_2010-Main. Accessed: 07-jun-2013.

OWASP (2013). "The Open Web Application Security Project". TOP 10 2013. Available: https://www.owasp.org/index.php/Top_10_2013-T10. Accessed: 07-jun-2013.

Sanches, B.; Basso, T.; Moraes, R. (2011) "J-SWFIT: A Java Software Fault Injection Tool". Fifth Latin-American Symposium on Dependable Computing - LADC. São Paulo, Brazil, pp.106-115.

Schneier, B. (1999) "Attack Trees: Modeling Security Threats", Dr. Dobb's Journal.

Uto, N., Melo, S.P. (2009). "Vulnerabilidades em Aplicações Web e Mecanismos de Proteção". Minicursos SBSeg 2009. IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2009.

 ©2013 SBC — Soc. Bras. de Computação