A Methodology for Effectiveness Analysis of Vulnerability Scanning Tools

Tania Basso, Regina L. O. Moraes (Co-orientadora), Mario Jino (Orientador)

Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101, CEP 13083-970 – Campinas, SP, Brasil

taniabasso@gmail.com, {regina@ft, jino@dca.fee}.unicamp.br

Abstract – Software systems developed nowadays are highly complex and subject to strict time constraints and are often deployed with critical software faults. In many cases, software faults are responsible for security vulnerabilities which are exploited by hackers. Automatic web vulnerability scanners can help to reveal these vulnerabilities. Trustworthiness of the results these tools provide is important; hence, relevance of the results must be assessed. We analyzed the effect on security vulnerabilities of Java software faults injected into source code of Web applications. We assessed how these faults affect the behavior of the vulnerability scanner tool, to validate the results of its application. Software fault injection techniques and attack trees models were used to support the experiments. The injected software faults influenced the application behavior and, consequently, the behavior of the scanner tool. A high percentage of uncovered vulnerabilities as well as of false positives points out the limitations of the tool.

Keywords – Fault injection, Vulnerability Scanner tool, Web Application Security.

1. Introduction

Web applications are extremely popular nowadays. This type of application is becoming increasingly exposed as any security vulnerability can be exploited by hackers.

Automatic vulnerability scanner tools are often used to assess Web applications with respect to 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. Previous research [1][2] shows that, in general, Web vulnerability scanners present a high number of false-positives (i.e., vulnerabilities detected by the tool that do not exist in the application) and low coverage (i.e., vulnerabilities that do exist in the application but were not identified by the tool), highlighting the limitations of this kind of tool.

Although other potential causes for vulnerability do exist, the root cause of most security attacks are vulnerabilities created by software faults [3][4]. Our proposal is to investigate the effect that Java software faults may have on security vulnerabilities and, then, analyze how they affect the behavior of the vulnerability scanner tool. The paper describes a method based on modelling of attack trees to define how to perform security tests by attacking the application.

The approach consists of injecting software faults into small Java applications to check if the scanner can detect potential vulnerabilities caused by the injected faults. Creation of vulnerabilities is confirmed through manual attacks, guided by the models of attack trees, to get accurate measures of detection coverage and false positives rate.

2. Software fault injection

Few works address the relationship between software faults and security vulnerabilities. A study by Fonseca and Vieira [5] analyzed security patches of web applications developed in PHP. The types of faults that are most likely to lead to security vulnerabilities are characterized.

The work by Basso *et al* [4] presents a field data study on real Java software faults, including security faults. The field study was based on security correction patches analysis available in open source repositories. More than 550 faults were analyzed and classified, determining the representativeness of these faults. The authors also define new operators, specific to this programming language structure, guiding the definition of a Java faultload.

The software fault injection technique used in this paper is the G-SWFIT [6], which is based on a set of fault injection operators that reproduce directly in the target executable code the instruction sequences that represent the most common types of high-level software faults.

To inject the faults, a use case of the application was selected. Each fault was injected in all possible locations of this specific use case,

one at time, forming different scenarios to be analyzed.

3. Effectiveness of vulnerability scanner tools

Web vulnerability scanners are regarded as an easy way to test applications against vulnerabilities. Most of these scanners are commercial tools (e.g., Acunetix [7], IBM Rational AppScan [8], N-Stalker [9] and HP WebInspect [10]).

Vieira *et al* [1] present an experimental evaluation of security vulnerabilities in publicly available web services. Four well known vulnerability scanners have been used to identify security flaws in web services implementations. Many differences in vulnerabilities were detected and a high number of false-positives and low coverage were observed when different tools were used to analyze the same application.

Fonseca *et al* [2] propose a method to evaluate and benchmark automatic Web vulnerability scanners using software fault injection techniques. Three leading commercial scanning tools were evaluated and the results have also shown that in general the coverage is low and the percentage of false positives is very high. However, these studies were focused on a specific family of applications: web services and PHP applications, respectively. Thus, the results obtained cannot be easily generalized. Furthermore, they do not present a clear methodology to validate the vulnerabilities detected by scanner tools. We investigate the behavior of scanner tools in the presence of injected Java faults, show a method using attack trees to model the possible ways to perform attacks to specific vulnerabilities, and analyze the results obtained by the scanner. This is addressed in the next sections.

4. Attack trees and security vulnerabilities

Attack trees provide a formal way of addressing security attacks on software systems [11]. In our work the attack trees are used to describe the various ways of attacking a specific type of security vulnerability. This is important to guide the security tests to validate the scanner results. We consider three types of security vulnerabilities: Cross-Site Scripting (XSS) [12], SQL Injection [13] and Cross-Site Request Forgery (CSRF) [14]. They were selected because they are widely spread and may cause major damage to the victims.

For each of these three types of vulnerability an attack tree was created. Figure 1 presents the attack tree for CSRF vulnerabilities. Due to space restrictions, the other trees are not presented, but they can be seen elsewhere [14].

In Figure 1, the first step to perform a CSRF attack is to have the user logged in the site because the attack will use the trust in user authentication. The next step is to analyze the request from the site that the attack will target in order to be able to reproduce it. If the site does not have CSRF countermeasures this step will lead to the next one because the request will be considered valid and will take effect on the site. If the site uses any defensive measure it will be necessary to analyze the request and take additional actions.

A known defensive method consists in appending different tokens to each request, but

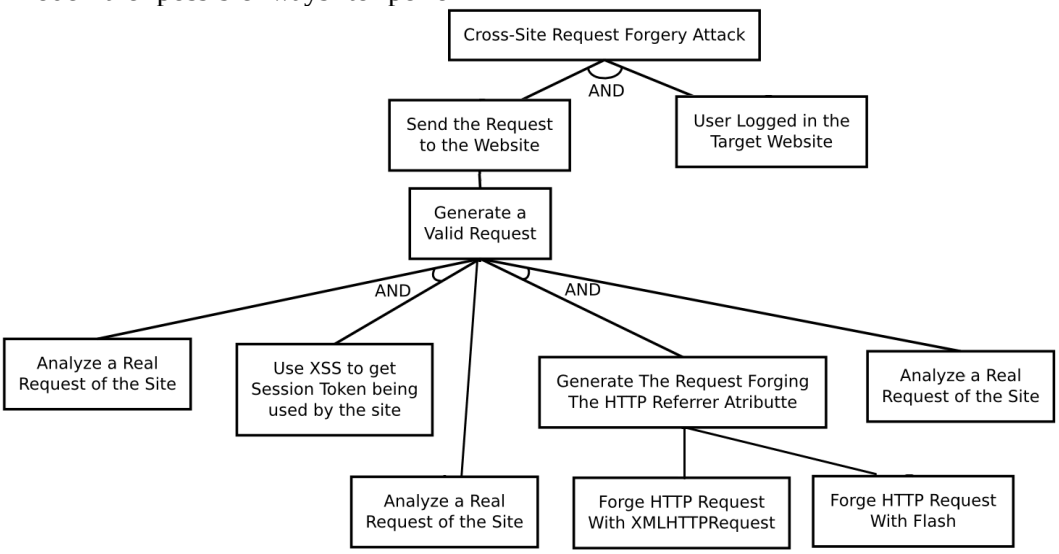

Figure 1. CSRF attack tree

this approach can be bypassed if the application is vulnerable to XSS attacks. The three remaining leaf nodes show how to overcome applications that use verification of the HTTP (Hypertext Transfer Protocol) Referrer attribute, although this is not a recommended defensive measure.

5. The experimental study

Two small open source Web applications developed in Java, App1 and App2, were selected to carry out the experiment. They are, respectively, a Customer Relationship Manager (CRM) and a management system for Distance Education, developed by the Brazilian federal government. They use technologies such as Hibernate and Ajax. We have chosen similar use cases from both applications to be the target piece of code of injected faults.

The types of fault to be injected were the most frequent ones observed by Basso et al. [4]. The security vulnerability scanner was selected because it is widely used and available. We do not identify it because commercial licenses do not allow the publication of tool evaluation results.

5.1 Injecting faults, executing the scans and validating the results

The tests start with a "Gold Run", where the application is tested once by the scanner tool without any fault injected. After the "Gold Run", one fault is injected. The context of the code where the fault is injected is analyzed to understand the effect of this fault in the applications behavior. Next, the code and the database are versioned, defining a scenario to be tested.

The scanner application is run and a verification of the results is done. If new vulnerabilities are detected, attacks are performed in the current scenario using the attack trees. This aims to verify if the new vulnerability actually exists or if it is a false positive. Then, the same attacks are performed in the original application scenario (without any fault injected) to verify if the vulnerability was present in the application before fault injection and was not identified by the tool (lack of coverage) when the Gold Run was performed.

The procedure is done for each possible location in the source code where faults can be injected in accordance with G-SWFIT technique (for the selected use case).

6. Results and discussions

For both Web applications, we analyzed, respectively, 11 and 23 different scenarios. Table 1 shows the total of scenarios that presented new security vulnerabilities detected by the scanner due to the fault injection.

Table 1. Applications scenarios and vulnerabilities

	App1	App2
Total scenarios analyzed	11	23
Scenarios with new vulnerabilities	5	7
% of faults that affected the scan	46%	30%

According to Table 1, about 35% of the injected software faults affected the scanner results. The lack of coverage and false positives rate are shown in Table 2.

In Table 2, the CSRF vulnerability represents 60% of the lack of coverage. In most of the cases, when scanning the application with fault injected, a new vulnerability detected by the tool was, in fact, one that already was present in the original application, not identified in the "Gold Run".

Table 2. Percentage of security vulnerabilities: lack of coverage and false positives

	XSS	SQL inject	CSRF	Total
Vulnerabilities	2	2	15	19
Lack of coverage (%)	0%	0%	60%	47%
False positive (%)	50%	100%	34%	42%

Also in Table 2, the false positives come from the three types of security vulnerabilities: XSS, SQL injection and CSRF, representing, respectively, 50%, 100% and 34% of the vulnerabilities detected. 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. Both applications use the Hibernate technology, and the way that the application was coded, i.e., extremely encapsulated, does not give opportunities to develop successful attacks.

Most of cases where CSRF false positives were identified were in error pages. A hacker

performing a CSRF attack to access an error page can be dangerous if the error page presents links or buttons which permit to access back the application (as "back" buttons which bring back the user to the last page he/she accessed). For both applications, the error pages do not present any way of accessing application functionalities or private information. Hence, we considered theses cases as false positives because a CSRF attack when accessing the error pages is useless.

The last column of Table 2 shows the total percentage of lack of coverage and false positives. From the 19 vulnerabilities investigated, 42% are false positives and 47% were not identified by the scanner tool. It indicates the limitations of this tool found in this study.

7. Conclusions

In this paper we present an experimental study where we analyze the effect of Java software faults, injected in the source code of Web applications, on security vulnerabilities. We also analyze the influence of these faults on the security vulnerabilities detection by a well known security vulnerability scanner tool. Fault injection techniques and attack tree modeling were used to support the experiments.

Results show that, according to the context of both the target code applications and the security vulnerabilities structure considered, the injected faults did affect the behavior of the application and, consequently, the behavior of the scanner tool in detecting new vulnerabilities. The scanner presented high percentage of lack of coverage and many false positives, showing its limitations. Factors that influenced this percentage are, in addition to the activation of the faults injected into the source code of the applications, the use of different development technologies (such as Hibernate) and some outdated features of the tool (as the internal internet browser).

We intend to extend this experiment by investigating the effect of other types of faults and the effectiveness of other vulnerability scanner tools. We also intend to develop a tool to perform the attacks (based on attack trees) automatically.

References

[1]M. Vieira, N. Antunes, H. Madeira. "Using Web Security Scanners to Detect Vulnerabilities in Web Services". *IEEE/IFIP Intl Conf. on Dependable Systems and Networks, DSN 2009*, Lisboa, Portugal, June 2009.

[2]J. Fonseca, M. Vieira, H. Madeira. "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks", *13º IEEE Pacific Rim Dependable Computing Conference (PRDC 2007)*, Melbourne, Victoria, Australia, December 2007.

[3]J. Fonseca, M. Vieira. "Mapping software faults with web security vulnerability". *IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, Anchorage, USA, 2008.

[4]T. Basso, R. Moraes, B. P. Sanches, M. Jino. "An Investigation of Java Faults Operators Derived from a Field Data Study on Java Software Faults*." In: Workshop de Testes e Tolerância a Falhas - WTF2009*, João Pessoa, Brazil, 2009, pp. 1-13.

[5]J. Fonseca, M. Vieira. "Mapping software faults with web security vulnerability". *IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, Anchorage, USA, 2008.

[6]J. Durães, H. Madeira. "Emulation of Software Faults: A Field Data Study and Practical Approach". *IEEE Trans. on Software Engineering*, Nov. 2006, pp.849-867.

[7]Acunetix Web Appication Security. Available in http://www.acunetix.com, November/2009.

[8]IBM Rational AppScan. Available in http://www01.ibm.com/software/awdtools/appscan/, November/2009.

[9]N-Stalker. Available in http://www.nstalker.com/, November/2009.

[10]HP WebInspect. Available in https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-201-200%5E9570_4000_100__, November/2009.

[11]B. Schneir. "Attack Trees: Modeling Security Threats", Dr. Dobb's Journal, December, 1999

[12]CGISecurity.com. "The Cross Site Scripting FAQ." Available in http://www.cgisecurity.com/xss-faq.html, November/2009.

[13]W. G. Halfond, J. Viegas, A. Orso, "A classification of SQL injection attacks and countermeasures". In Proc.IEEE International Symposium on Secure Software Engineering, Arlington, Virginia, March/2006.

[14]R. Auger. "The Cross-Site Request Forgery (CSRF/XSRF) FAQ". Available in http://www.cgisecurity.com/csrf-faq.html, November/2009.

[15]Research Test Group. Available in http://www.ceset.unicamp.br/docentes/regina/projeto/, December/2009.