# ETSSDetector: a tool to automatically detect Cross-Site Scripting vulnerabilities

Thiago S. Rocha and Eduardo Souto
Institute of Computing
Federal University of Amazonas
Manaus, Amazonas
{thiago.rocha,esouto}@icomp.ufam.edu.br

***Abstract*— The inappropriate use of features intended to improve usability and interactivity of web applications has resulted in the emergence of various threats, including Cross-Site Scripting (XSS) attacks. In this work, we developed ETSSDetector, a generic and modular web vulnerability scanner that automatically analyzes web applications to find XSS vulnerabilities. ETSSDetector is able to identify and analyze all data entry points of the application and generate specific code injection tests for each one. The results shows that the correct filling of the input fields with only valid information ensures a better effectiveness of the tests, increasing the detection rate of XSS attacks.**

***Keywords—Cross-Site Scripting, ETSSDetector, vulnerabilities.***

## I. INTRODUCTION

Web applications have been used as one of the most important channel of communication between service providers and users. Therefore, ensure the safety of these applications became a priority and indispensable task for application developers.

The most common web application security weakness is the failure to properly validate input from the client or environment. This weakness leads to the most common vulnerability in applications, the injection of malicious code also known as Cross-Site Scripting (XSS) [1].

Numerous approaches have been used as countermeasures for XSS prevention in "real-life" web applications, including quotes static analysis [2], white [3] and black-box [4] tests, and anomaly detection [5]. Regardless of the approach chosen, the black-box vulnerability scanners are one of the main tools used by security professionals to run vulnerability tests on web applications.

This choice is justified by the set of features that these tools should provide as ease of use, fast implementation of the tests, and ability to identify a large number of vulnerabilities. However, recent researches shown that many of these tools have difficulty dealing with different levels of complexity employed by different attacks. Bau, Bursztein, Gupta and Mitchell [6] present an analysis about a set of commercial vulnerabilities scanners and concluded that the stored XSS detection rate was only 15%. Similar conclusions about the effectiveness of these tools also were taken on [7]. McAllister, Kirda and Kruegel [8] also observed that the scanners often fail to test a substantial fraction of a web application's logic, especially when this logic is invoked from pages that can only be reached after filling out complex forms. In other study, Kosuga Y. [9] reports that the majority of the vulnerabilities scanners apply all possible tests on all input fields of the application, creating meaningless attacks that could never be activated.

To deal with these weaknesses, this paper presents ETSSDetector, a tool developed to automatically detect XSS vulnerabilities, through the analysis of the information contained on web applications. ETSSDetector is built on techniques that enable the correct filling of form fields with valid information allowing the pages to be successfully submitted. The qualification of these fields allows the tool to obtain all the pages of the application, enabling a broader analysis and increasing the detection rate of XSS attacks.

The main contributions of this paper are as follows:

- We show how easy it is for attackers to execute XSS exploits and that the efficiency of vulnerability scanners needs to be improved.
- We developed a form field qualification module aiming to guarantee the submission of complex forms, treating the problem identified in [8].
- We propose a qualified attack injection, where the tests are made based on the kind of vulnerable point being tested, increasing the effectiveness of the tool and treating the problem appointed in [9].

The remainder of this paper is organized as follows. Section 2 describes our approach for automated vulnerability detection. Section 3 gives an experimental evaluation of our tool. Finally, Section 4 concludes the paper.

## II. ETSSDETECTOR

When doing XSS vulnerability tests on web applications, some problems need to be focused: 1) How to identify all the vulnerable points? 2) How to access page forms and fill them? 3) How to collect all web application pages? 4) Which attacks use in each vulnerable point? 5) How to identify if an attack was executed correctly?

To deal with these questions, this work proposes a tool to test XSS web application vulnerabilities called ETSSDetector. This tool uses some techniques (described on following

978-1-4799-5393-6/14 $31.00 © 2014 IEEE
DOI 10.1109/NCA.2014.53

sections) that make possible the fill of each vulnerable point with valid values allowing the correct submission of pages. Figure 1 shows how ETSSDetector process works.

### A. Emulation

In order to detect XSS vulnerabilities in a given Web application, ETSSDetector should be able to simulate a browser's behavior, i.e. loads the page and executes its dynamic content as clicking in links or filling out forms fields. To meet these functionalities, a full browser environment is emulated by HtmlUnit, a Java-based framework for testing web-based applications [10]. We have decided to use HtmlUnit rather than instrumenting a traditional browser, such as Firefox or Internet Explorer, because there exists some attacks that work only on a specific browser and with HtmlUnit it is possible to simulate multiple browser personalities.

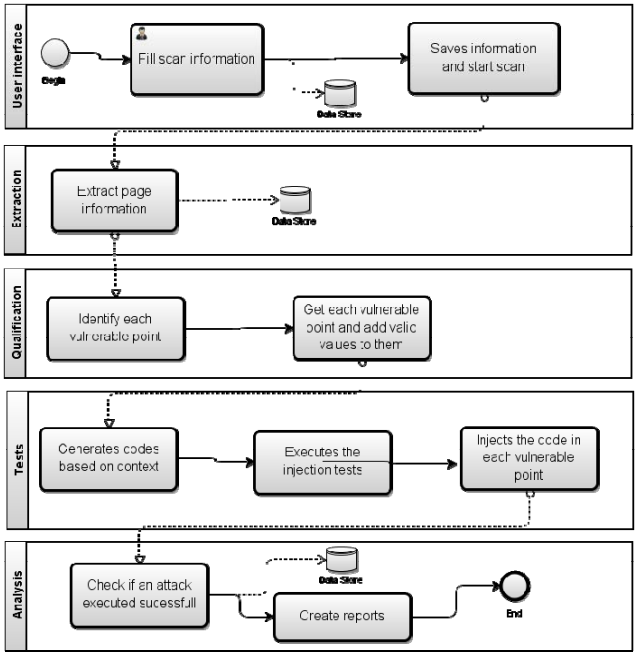


Figure 1. ETSSDetector components.

### B. Extraction Component

Extraction component is responsible to identify, collect and analyze the necessary information to evaluate the web application. Extraction process collects information like links and forms available, form parameters, as well as identifies which method, GET or POST, the forms will execute.

As shown in Figure 1, the process starts with a page to be visited. The extractor makes a request to a HTTP server, downloads the document and begins to extract links, content and other information. Link addresses are saved so they can be accessed later.

Like most of existent page collectors, the process forecasts two types of HTTP requests, GET and POST. In case the request is made via GET, page address parameters are extracted and inserted on database. In case the request is made via POST, form parameters are recovered and inserted on database. During this process new pages can be found and must pass through the same process.

### C. Qualification Component

This component makes the analysis of the pages of the web application being tested identifying each vulnerable point. The goal of this process is to identify which values are more appropriated to fill each vulnerable point of a specific form.

This component checks if there is a qualifier at the database that matches with the value of the vulnerable point. If it exists, the value of the qualifier is assigned to the vulnerable point being qualified. Otherwise, the process keeps searching the values of attributes and following elements in a DOM tree. In case there is not a qualifier that matches these values the process keeps looking for the element on the right side of the point. Finally, if no qualifier is found the field receives a standard value.

This module presents a difference from other tools because the qualification module aims to fill each field with valid values only, ensuring the submission of complex forms, mitigating the problem found by [8].

### D. Test Component

The test component is responsible to inject and execute XSS attacks that will be assigned to each vulnerable point. Differently from most vulnerability test tools, ETSSDetector proposes the use of a customized attack selection process, i.e., tests are made based on the kind of vulnerable point being tested. For example, if the vulnerable point is at the title of the page, only tests related to title will be used. The main goal of this approach is to avoid the execution of a big number of unnecessary attacks, solving the problem identified in [9]. This process starts checking if the page generates a GET or POST request. Based on this information the parameters are recovered from the database. Next step is the injection of the malicious codes in each vulnerable point of the page being tested.

Before inserting the real malicious code, a simple injection test is added to each vulnerable point to determine the position of the points in the response page. This information allows the code generator to create only appropriate attacks to each vulnerable points, this approach is called "dummy injection code" and is also used on WebXSSDetector tool [11].

### E. Analysis component

This component performs the detection of XSS attacks through the analysis of the results from previous modules, with these information ETSSDetector check if the attack is executed or not and after creates reports of the application being tested.

The differential of this module is because ETSSDetector performs the detection of the attack in the whole application. Most of other tools check only the directly response created by the request, with this approach ETSSDetector aims to increase the detection percentage of persistent XSS attacks, problem found on [9].

The process starts getting the list of tests created on the previous module. Iteration occurs at this list to insert each test at a vulnerable point. Next step is to execute the test to verify its result, if it is positive the database is updated, otherwise, the iteration continues until the list is empty.

### *F. Database*

All information collected during ETSSDetector execution are stored in a database like links, forms and parameters. The use of a persistent repository turns possible the access to the information in any time.

## III. EVALUATION

This section demonstrates the effectiveness of ETSSDetector on vulnerabilities reported at the real and experimental scenarios. We compare ETSSDetector with others six web vulnerability scanners using three different applications.

ETSSDetector evaluation results were compared with Acunetix [12], N-Stalker [13], WebCruiser [14], PowerFuzzer [15], WebSecurify [16], and WebXSSDetector[11]. The criteria to select these tools were the availability to access the tool, goals similarity and the perception of use of the tool on the market. In the set of tools Acunetix is considered one of the best vulnerability scanners in the market according to the studies cited in [6] and [7]. However, the free version of Acunetix that is available for tests has some limitations, which allowed the use of it only in the first test scenario.

The metrics used to evaluate the tools were: *i)* amount of XSS attacks detected; *ii)* quantity of tests generated at the application; and *iii)* runtime of the tools. These metrics were chosen because they are used in previous researches and because they verify the resolution of the problems mentioned in the introduction.

Data collection for analysis occurred through the analysis of the behavior of the tools, reports generated and, when possible, from logs from the server where the application was stored.

### *A. First evaluation scenario*

First evaluation scenario used an application, available in *testphp.vulnweb.com*, developed by Acunetix owners. Table 1 shows the results of the number of attacks found.

Table 1. Execution time of web vulnerabilities scanners.

| ***Tool*** | ***Attacks XSS found*** | ***Average Scan Time*** |
|---|---|---|
| Acunetix | 17 attacks | 2 minutes and 2 seconds |
| PowerFuzzer | 12 attacks | 14 minutes and 51 seconds |
| N-Stalker | 9 attacks | 5 minutes |
| WebSecurify | 4 attacks | 37 seconds |
| WebXSSDetector | 15 attacks | 4 minutes and 9 seconds |
| WebCruiser | 8 attacks | 2 minutes and 30 seconds |
| ETSSDetector | 17 attacks | 2 minutes and 37 seconds |

The first point analyzed on this scenario is that most of the scanners, including ETSSDetector, got an execution time between 2 and 5 minutes, except PowerFuzzer tool that got an execution time of 14 minutes and WebSecurify tool that executed in 37 seconds.

Regarding XSS attacks, Acunetix and ETSSDetector detected the largest number of vulnerabilities, 17 each, 15 of them being the same. The logs analysis shows that Acunetix tool found two attacks that are executed through cookies, functionality that ETSSDetector tool does not have. However, ETSSDetector found two others attacks that Acunetix could not detect. Only ETSSDetector and WebXSSDetector found it.

These attacks occur on the page that is used to create accounts *http://testphp.vulnweb.com/signup.php*. On them an attacker uses the fields "*password*" and "*retype password*" to insert malicious code. The code inserted must be the same in both fields, as both are vulnerable ETSSDetector detected two vulnerabilities. To prove that these attacks are not false positives they were manually reproduced successfully.

### *B. Second evaluation scenario*

On second scenario it was developed an application with vulnerabilities to test the qualification process designed by ETSSDetector. This application has four fields: name, surname, email and ID.

Two of these fields have vulnerabilities (*name* and *surname*) and other two needs a validation to submit the form (*email* and ID). Purposely the field *name* is displayed on the title of the response page and *surname* is displayed as an attribute of a HTML tag.

This proceeding was made in order to test if ETSSDetector generates less tests (or requests) to each vulnerable point through the application of test injection technique. Table 2 shows the results.

Table 2. Number of requests generated by tools.

| ***Tools*** | ***Request Number*** | ***Attacks XSS found*** |
|---|---|---|
| PowerFuzzer | 64 requests | None |
| WebSecurify | 37 requests | All |
| WebXSSDetector | Runtime exception | None |
| WebCruiser | 60 requests | All |
| ETSSDetector | 15 requests | All |

Table 2 shows that between the scanners that detected the attacks, WebCruiser generated more than four times the number of requests ETSSDetector made and WebSecurify more than double. This difference is due to the ETSSDetector test component, responsible to inject and execute XSS attacks that will be assigned to each vulnerable point. This component verifies the response page and all collected pages, so that information can be obtained in order to generate appropriate injection codes.

The importance the appropriate injection codes can be comproved by results of PowerFuzzer tool. This tool made the highest number of requests, 64 requests in total. However, it

passed invalid values in all the requests making the XSS attacks undetectable, as depicted in the PowerFuzzer log in Figure 2.

The PowerFuzzer tool passed Google address to the email field and the text "*on*" to the ID field, making the form invalid.

```
http://localhost/XSS/submitpost.php/post.php
Attacking forms (POST)...
+ http://localhost/XSS/submitpost.php/post.php
'surname' => 'on'
'Email' => 'http://www.google.com/'
'id' => 'on'
'name' => 'on'
```

Figure 2. PowerFuzzer log.

### C. *Third evaluation scenario*

On this scenario were conducted tests on an application that does room reserves, available in *http://www.webscantest.com/crosstraining/reservation.php*. This page has two XSS vulnerabilities, both non-persistent. However, one of them behaves as it was persistent, because it is not immediately reflected to the user.

Table 3 shows that only ETSSDetector was capable to detect both vulnerabilities. WebCruiser detected one and the others did not find any. In order to verify if the attack is a false positive it was manually reproduced.

Table 3. Number of attacks detected.

| ***Tools*** | ***Number of attacks detected*** |
|---|---|
| PowerFuzzer | None |
| WebSecurify | None |
| WebXSSDetector | None |
| WebCruiser | One |
| ETSSDetector | Two |

Test analysis on ETSSDetector is not made based on the direct response from a request, it checks all the pages of the application. This happened in an attempt to detect both persistent and non-persistent XSS attacks, this scenario shows that the attempt was successful because only ETSSDetector could find both vulnerabilities.

## IV. CONCLUSION

In this work we have designed ETSSDetector, a web scanning tool to help testers and developers automatically detect XSS vulnerabilities in web applications.

The results show effectiveness of using ETSSDetector in identification of the actual and hidden vulnerabilities by analyzing each vulnerability point in the web application. The test injection technique employed in the tool reduced the amount of tests generated on each vulnerable point and the use of field qualifiers increased the assurance of the submission of complex forms.

The verification of the tests in all the pages of the application, together with the test injection and qualification techniques raised the rate detection of XSS attacks, as shown in the comparisons with other tools. Thus, ETSSDetector is presented as a good alternative to assist information security professionals in the fight against the vulnerabilities that are presented in web systems.

As future work we intend to change the test module to enable that users add new attacks. Another important feature would be the evaluation of ETSSDetector to add mechanisms to detect XSS DOM failures. Finally, we aim to extend the use of the proposed tool adding mechanisms to detect other vulnerabilities in web applications such as SQL Injection.

## V. REFERENCES


[1] Grossman, J., Hansen, R., Petkov, P., Rager, A. and Fogie, S. "Cross site scripting attacks:XSS Exploits and defense.", Syngress,Elsevier, pages 67-179, 2007.

[2] Jovanovic N., Kruegel C., Kirda E. "Pixy : A Static Analysis Tool for Detecting Web Application Vulnerabilities", IEEE Symposium on Security and Privacy, pages 258-263, 2006.

[3] Tsai D., Chang A., Liu P., Chen H. "Optimun Tuning of Defense Settings for Common Attacks on the Web Applications," IEEE Carnahan Conference, pages 89-94, 2009.

[4] Rawat, S., Richier, J.-L., Groz, R. "LigRE: Reverse-engineering of control and data flow models for black-box XSS detection", IEEE Reverse Enginering Working Conference, pages 252-261, 2013.

[5] Shanmugam, J., Ponnavaikko, M. "Behavior-Based Anomaly Detection on the Server Side to Reduce the Effectiveness of Cross Site Scripting Vulnerabilities" IEEE Semantics, Knowledge and Grid Conference, pages 350-353, 2007.

[6] Bau, J., Bursztein, E., Gupta, D. and Mitchell, J. "State of the Art: Automated Black-Box Web Application Vulnerabilitty Testing," IEEE Symposium on Security and Privacy Vulnerability Testing, pages 2-5, 2010.

[7] Doupe, A., Cova, M. and Vigna, G. "Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners," Seventh Conference on Detection of Intrusions and Malware and Vulnerability Assessment, pages 1-10, 2010.

[8] McAllister, S., Kirda, E. and Kruegel, C. "Leveraging User Interactions for In-Depth Testing of Web Applications," ACM international symposium on Recent Advances in Intrusion Detection, pages 1-2, 2008.

[9] Kosuga, Y."A Study on Dynamic Detection of Web Application Vulnerabilities" Ph.D. dissertation, School of Science for Open and Environmental Systems, Keio, Japan, 2011

[10] Gargoyle. "HtmlUnit - Welcome to HtmlUnit," Available: http://htmlunit.sourceforge.net/, January 2014.

[11] Jia, X. "Design, Implementation and Evaluation of an Automated Testing Tool for Cross-Site Scripting Vulnerabilities," Diploma thesis, Darmstadt University of Technology, Darmstadt, Germany, 2006.

[12] Acunetix. "Website Security with Acunetix Web Vulnerability Scanner," Available: http://www.acunetix.com/, January 2014.

[13] N-Stalker. "N-Stalker The Web Security Specialists," Available: http://www.nstalker.com/, January 2014.

[14] WebCruiser. "Download WebCruiser," Available: http://sec4app.com/download.htm, February 2014.

[15] PowerFuzzer. "PowerFuzzer - a fuzzer that introduces powerfull and easy web fuzzing," Available: http://www.powerfuzzer.com/, February 2014.

[16] WebSecurify. "WebSecurify Online Web Application Security Scanner and Web Security Testing Tool," Available: http://www.websecurify.com/, March 2014.