Available at: www.sabauni.net/ojs

S.J.I.T.N
Saba Journal of Information Technology and Networking

PALXSS: CLIENT SIDE SECURE TOOL TO DETECT XSS ATTACKS

Tawfiq S. Barhoom *, Mohammed H. Abu Hamada

Islamic University Gaza, Palestinian Territory, Occupied

Article info

Article history: Accepted May ,2014

Keywords:

Cross-Site Scripting
Malicious script codes
Client side

Abstract

Cross-Site Scripting is one of the main attacks of many Web-based services. Since Web browsers support the execution of scripting commands embedded in the retrieved content, Attacker can gain this feature maliciously to violate the client security such as confidentiality. The public sites (i.e. social network) provide the attacker with ability to post there malicious code into a context which in the future to be shown to other participants. Detecting these malicious script codes is necessary for client side; the detection can be done by using detection tools used at client side. This paper describes the overall problem and elaborates on the possibilities to solve the problem with actions at client side to reduce the danger of Cross-Site Scripting attacks. In this work a new secure tool is developed using python language, which called PalXSS, two factors are used to evaluate it: performance and accuracy. The results show the accuracy of PalXSS tool is 90.24% which satisfies the users need compared with other tools.

* Corresponding author: Dr. Tawfiq S. Barhoom.
E-mail address: tbarhoom@iugaza.edu.ps

Introduction

Social networks, such as Facebook and MySpace, blogs and micro-blogs, such as Twitter, and other content providing services that are built on users' collaboration, such as YouTube and Flickr, are considered the killer applications of the last few years [1]. Also, everything has two sides. On the opposite side, these dynamic websites also provide a good platform for hackers to inject malicious code, as well. If the code is executed behind the web browser, it changes the web page according to the code automatically. Therefore, a lot of famous websites were injected with malicious code by hackers and a lot of visitors were attacked. Moreover, owing to the extensive spread of Web 2.0 and each user's blog can be shared with his/her friends as well. So, if one blog has been injected with malicious code, all the visitors of the blogger's friends will be infected and constantly infect their friends. Therefore, the speed of spreading is even quicker than previously. Eventually, the website provider will lose a lot of money and its reputation will be damaged, as well [2].

Cross-site scripting (XSS) attack method was first discussed in Computer Emergency Response Team (CERT) advisory back in 2000 [3]. But, even today cross-site scripting is one of the most common vulnerabilities in web applications; it has a widespread vulnerability in Web applications and was ranked first in OWASP Top Ten report 2007 and second in OWASP Top Ten report 2010 [4]. It happens as a result of data received from a malicious person and then sent to third parties. Systems that receive data from users and display it on other users' browsers are very vulnerable to an XSS attack. Wikis, forums, chats, web mail - are all good examples of applications most susceptible to XSS. This type of vulnerability allow hackers to inject the code into the output application of web page which will be sent to a visitor's web browser and then, the code which was injected will execute automatically or steal the sensitive information from the visits input. This code injection, which is similar to SQL Injection in Web Application Security,

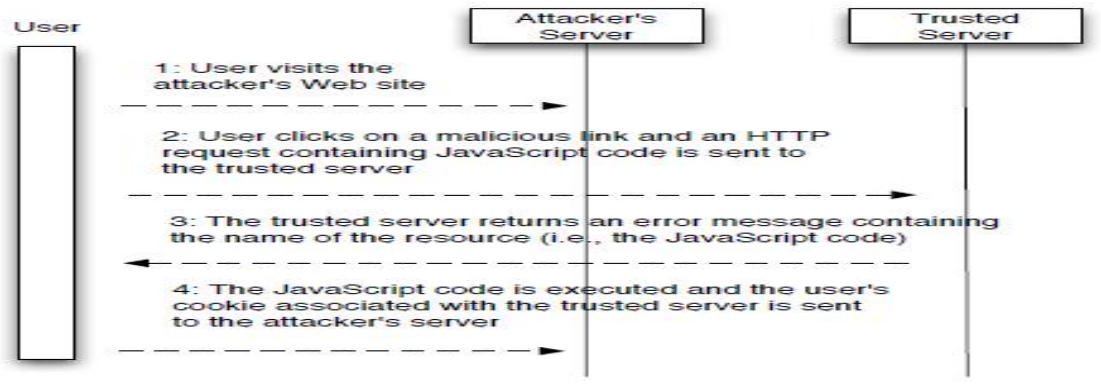

Fig.1. A typical cross-site scripting scenario [5].

can be used in three different ways namely "Persistent XSS", "Non-Persistent XSS" and "Dom- based XSS" Fig. 1. Vulnerabilities in Web applications can be discovered in various ways. In the black-box approach the Web Vulnerability Scanner has no knowledge about internal operation and operates only on the interfaces that can be accessed from the outside. The internals of the application are kept secret, source code cannot be accessed and most of the time, the Web Vulnerability Scanner doesn't even know which type of Web server the application runs on. All information about the Web application must be gathered with the help of tools such as Web Vulnerability Scanners or manually by inspecting the HTTP responses and by trying different input values to understand the behavior of the Web application [6]. In white-box testing [6], the opposite is true. The Web Vulnerability Scanner has access to the internal workings of the Web application and every request can be traced. All necessary information is then available and can even access the source code to find vulnerabilities. The internal mechanisms of the Web application can be traced in detail using debugging tools.

In the scope of this work, only black-box techniques are investigated as black-box testing is typically the case for most Web Vulnerability Scanners testers and also for attackers with malicious intent. To find the vulnerability, python language was used which is simple language, an easy to learn, powerful programming language and free and open source language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms [7].

Related Works

There are largely two distinct countermeasures for XSS prevention at server side: Input filtering and output sanitation. Input filtering describes the process of validating all incoming data. The protection approach implemented by these filters relies on removing predefined keywords, such as <script, JavaScript, or document. Output sanitation is employed, certain characters, such as <, ", or ', are HTML encoded before user-supplied data is inserted into the outgoing HTML. As long as all untrusted data is "disarmed" this way, XSS can be prevented. Both of the above protections are known to frequently fail [8], either through erroneous implementation, or because they are not applied to the complete set of user-supplied data. Client side solution acts as a web proxy to mitigate Cross Site Scripting attack which

manually generates rules to mitigate Cross Site Scripting attempts. Client side solution effectively protect against information leakage from the user's environment. However, none of the solutions satisfy the need of the client side. There are several client-side solutions.

Hallaraker et al. [9] proposed a strictly client-side mechanism for detecting malicious JavaScript's. The system uses an auditing system in the Mozilla Firefox web browser that can perform both anomaly or misuse detection. This system monitors the execution of JavaScript and compares it to high level policies to detect malicious behavior. This solution is insufficient because if new vulnerabilities should be detected, new rules have to be implemented and the browser has to be rebuilt. Also it is possible to detect various kinds of malicious scripts, not only XSS attacks. However, for each type of attack a signature must be crafted, meaning that the system is defeated by original attacks not anticipated by the signature authors. Some authors [10-14] have proposed the use of static analysis techniques to discover input validation flaws in a web application; however, this approach requires access to the source code of the application [10, 11]. Moreover, those static analysis schemas are usually complemented by the use of dynamic analysis techniques. Huang et al [12], Balzarotti et al [14] used this techniques to confirm potential vulnerabilities detected during the static analysis by watching the behavior of the application at runtime. Several existing systems have been adapted to detect XSS. Application level firewalls [5], reversal proxies [15] and IDS (Intrusion detection systems) [16, 17], have been adapted to try to mitigate the XSS problem. Firewalls focus on tracking sensitive information and controlling whenever data is to be sent to untrusted domains. Reverse proxies receive all responses from the web application and check whether there are any unauthorized scripts on them. IDS approaches deal with the identification of traffic patterns that allow the detection of known XSS attacks.

Kirda et al [5] presented Noxes as a client-side Web-proxy that relays all Web traffic and serves as an application-level firewall. The main contribution of Noxes is that it is the first client-side solution that provides XSS protection without relying on the web application providers. Noxes supports an XSS mitigation mode that significantly reduces the number of connection alert prompts while at the same time providing protection against XSS attacks where the attackers may target sensitive information such as cookies and session IDs. The approach works without attack-specific signatures. The main problem of Noxes as that it requires user-specific configuration (firewall rules), as well as user interaction when a suspicious event occurs.

Another client-side approach was presented by Vogt et al [13], which aims to identify

information leakage using tainting of input data in the browser. The solution presented in this paper stops XSS attacks on the client side by tracking the flow of sensitive information inside the web browser. If sensitive information is about to be transferred to a third party, the user can decide if this should be permitted or not. As a result, the user has an additional protection layer when surfing the web, without solely depending on the security of the web application.

Gal´an et al [18] completed the scope of vulnerability scanners by allowing them to check the presence of stored–XSS vulnerabilities in web applications. The system proposed was based on multi–agent architecture allowing for each one of those tasks to be carried out by a different type of agent. This design decision has been taken to allow each of the stages of the scanning process to be performed concurrently with the other stages. It also allows for the different subtasks of the scanning process to take place in a distributed and/or parallel way. The agent that explores the web site in order to find the injection points where stored–XSS attacks could be launched. This parsing process is similar to that of web crawlers and spiders. XSS-Me the One of the best open source tools was the Exploit-Me series presented by securitycompass.com [19]. Security Compass created these tools to help developers easily identify cross site scripting (XSS) and SQL injection vulnerabilities.

XSS-Me is a Firefox add-on that loads in the sidebar. It identifies all the input fields on a page and iterates through a user provided list of XSS strings: opening new tabs and checking the results. When this process completes you get a report of what attacks got through, what didn't, and what might have. The tool does not attempt to compromise the security of the given system. It looks for possible entry points for an attack against the system. There is no port scanning, packet sniffing, password hacking or firewall attacks done by the tool. You can think of the work done by the tool as the same as the manual testers for the site manually entering all of these strings into the form fields. This tool is good for detecting XSS attacks but it needs user interaction to do testing (like manual testing), moreover its cannot follow all links in the website, as a result, it scans the link provided by the user click.

All client-side solutions share one drawback: the necessity to install updates or additional components needed on each user's workstation. While this might be a realistic precondition for skilled, security-aware computer users, it is perceived as an obstacle or is not even considered by the vast majority of users. Thus, the level of protection such a system can offer is severely limited in practice.

Methodology

Current fully automated Web Vulnerability Scanners (WVS) has three major components: A crawling component, an attack component and an analysis component [20]:

*1. **Crawling Component**:*

The crawling component collects all pages of a Web application. It uses an input URL as seed starts following links on each page and store the result in list. The crawling module is arguably the most important part of a Web application Vulnerability Scanner; if the scanner's attack engine is poor, it might miss vulnerability, but if it's crawling engine is poor, then it will surely miss the vulnerability [21].

2. Attack Component:

The attack component scans website, extracts all internal links then scans all crawled pages forms field which are used in URL parameters then injects various attack patterns into these parameters; Parameters can be part of the URL query string or part of the request body in HTTP POST requests. Both are equally exploitable. In this work, most examples have forms with input fields to illustrate vulnerable parameters [20].

3. Analysis Component:

The analysis component parses and interprets the server's responses. It uses attack-specific criteria and keywords to determine if an attack was successful. An attack vector is a piece of HTML or JavaScript code that is added into a parameter in-order to be reflected to users by being embedded into a HTTP response. The goal of an attack vector is to make user browser execute malicious code. The malicious code can be either fetched from trusted website or be part of the attack vector itself, although the former allows more complex exploits. two examples for typical attack vectors are:

1. <script src="http://attacker.com/exploit.js"></script>

loads and executes a remote script from website.

2. <body onload="document.write('<img src=http://attacker.com/?'+document.cookie+'/ >')">

performs cookie stealing as part of the attack vector.

Our proposed model architecture is shown in fig. 2. In step 1, all pages are crawled and stored into the list (step 2). For simplicity and easy installation, data is stored in a text file rather than in a database. Stores are only small amounts of data (a few kilobytes) that don't cost much performance. In step 3, the attack module takes pages from the list with modifiable parameters, injects attack vectors and passes the responses to the analyzer, which analyzes them for injected patterns in step 4. In this step, the attack component injects a

common attack vector such as <script>alert (“XSS”) </script> and the analysis component uses a regular expression to search for the very same injection string. If the attack pattern is found unmodified (no characters were added or replaced), the attacked parameter is vulnerable to XSS.

PalXSS tool are used to detect XSS attack by performing an attack and checking the resulting page if the malicious code is injected without modification. The steps are:

1. A selection of attack vectors is obtained from an attack vector repository; XSS attack vectors are commonly stored in repositories and include the description of the attack as well as the script code to be injected.

2. Selected attack vectors are launched against inputs of the web application. Those attack vectors are generally injected in an

HTTP request as parameters or as fields in a web form.

3. PalXSS tool receives the responses to the requests containing the injected code.

4. The PalXSS tool checks for the presence of injected script in the received responses. If affirmative, XSS attack is considered successful and a vulnerability of the scanned web application has been discovered.

Implementation

PalXSS is a secure tool which is written in python language. The tool consists of four main classes; these classes are:

1. ***Web Page Parser class:***

When the client launches this class, python script will prompt him to enter a URL. The script will connect to the URL entered and hunt for any <a href> elements, as it systematically retrieves information from the pages it visits and it propagates through the site following the hyper-links it finds. Nevertheless, it differs from the typical web crawler in two aspects: (1) It just follows the hyper-links with destination to the scanned site discarding all external links and, (2) the information recovered are web forms

2. ***Spider Class:***

In this class, the script will connect to the URL entered in the previous step and hunt for any <form> elements. It will output the attributes associated with the elements allowing client to see what method is being used and what action is being performed. Once all the <form> elements are collected, it will then move on to <input> tags. All entries found will then be displayed as "possible" targets.

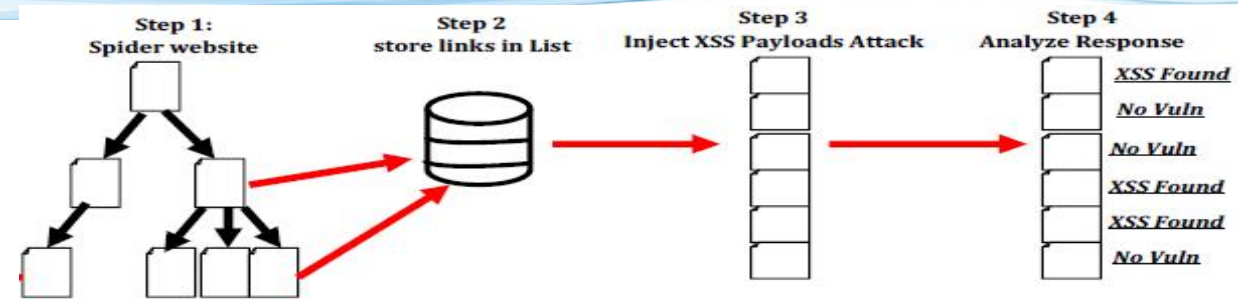

Fig 2: Proposed model architecture

3. *Script Injector Class:*

This class extracts the collection of web forms elaborated by the web page parser class and register in the injection repository. The class will inject a collection of XSS attack vectors from a well-known repository into different input fields of each of the injection points

4. *The Store Class:*

This class shows the report which contains: the links extracted from the base URL, and the input form field hacked. This report helps the client to know the XSS vulnerable in the website.

Dataset

We performed a series of experiments with our prototype implementation to demonstrate its ability to detect previously known cross-site scripting vulnerabilities, as well as new ones. To this end, PalXSS was run on seven popular XSS Payloads. The dataset of attacks used for evaluation our tool were extracted from a repository of XSS attack vectors found in http://ha.ckers.org/xss.html. Those vectors use different ways of inserting arbitrary script code trying to be unnoticed by the web application and, in our case, to be incorporated as legitimate content in the web application. As the attack vectors in the repository are large, the experiment tests every type to define the code accepted. The XSS payloads show the code accepted by testing our tool in real websites. The number of injection attacks can affect the performance of detection. To enhance the performance, we took seven attacks as default in our tool that was accepted in most of the tests.

Evaluation

This section presents the evaluation of PalXSS tool. The task was to detect all XSS vulnerabilities in online website. Different categories of tests were conducted to ensure that our solution works. The two major aspects of the evaluation application are (i) to compare our work architecture with the traditional architecture of scanners and (ii) the comparison of the execution time and accuracy by three tools.

An implementation of the proposed system was developed with the purpose of testing and evaluating the scanner against different websites; three scanners were used for the evaluation. These scanners work at the same condition with the same parameters; also these tools share the same methodology. The tools are:

Acunetix 7, XSSploit: and PalXSS. Fig. 3 shows the execution time of our tool compared to other tools e.g., the first website of testing is http: xss.progphp.com; the execution time of our tool is 84/sec, it has a better performance compared to XSSploit tool, while it has law performance when compared to Acunetix tool.

The execution time of our tool is the minimum in all cases than other tools, while in some cases such as in website 8 as shown in fig. 3 the execution time is the maximum, this result occurs because the number of field detected in this site is ten which takes more time to check the result than other tools which can't detect them, this gives the best accuracy.

The result in the table 1 shows the accuracy of PalXSS tool as the best compared to other tools; the average detection rate of PalXSS tool even 90.24%, while the average detection rate of XSSploit was 24.39% and the average detection rate of Acunetix tool was 57.32%.

The accuracy of PalXSS tool can be satisfying the users to use this tool among others.

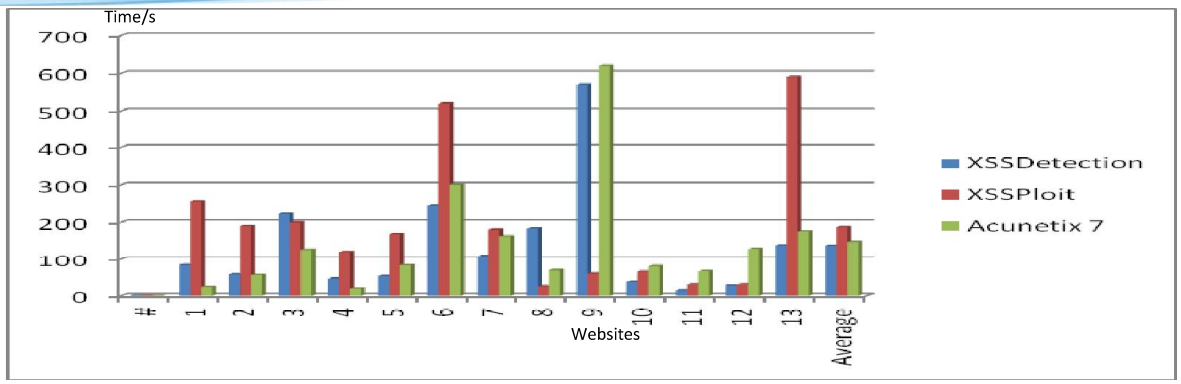

Fig.3. Comparison of three tools in this works

#	Websites	Vuln. Filed	Pal XSS	XSS Sploit	Acunetix
			Vulnerable field Detected		
1	http://xss.progphp.com	2	2	2	2
2	http://testasp.vulnweb.com	1	1	1	1
3	http://demo.testfire.net	2	1	2	2
4	http://www.kaspersky.com.pt/base/guest/mimemessage/test_multibyte_message.php	6	6	0	0
5	http://testphp.vulnweb.com	2	0	2	2
6	http://demo.arcticissuetracker.com	2	1	0	2
7	http://zero.webappsecurity.com	5	1	5	5
8	http://www.binaryanalysis.org/en/home	10	10	1	6
9	http://www.socialweb.net/Accounts/general.lasso?new=1	25	25	0	10
10	http://www.qou.edu/contactUs.do?key=2	6	6	5	2
11	http://www.maktoobblog.com/search	1	1	1	1
12	http://www.gametiger.net	5	5	1	5
13	http://www.asianave.com/user/register.html	15	15	0	9
Total		82	74	20	47
Average			**90.24%**	**24.39 %**	**57.31%**

Table 1: the detection rate of three tools

Recommendations

We recommend that the regular security tests need to be part of an effective software development process; furthermore, detected tool must play an important role in providing a testing framework. The developers must train well enough about the security holes in the website. Security awareness and education is incorporated throughout several stages such as creating documentation, threat modeling etc. Nevertheless, it is important to understand that the goal of vulnerability scanning is to reveal security flaws so that developers can trace these issues and implement security mechanisms. In addition, we propose that as our culture becomes more dependent on information, social engineering will remain the greatest threat to any security system. Prevention includes educating people about the value of information, training them to protect it, and increasing people's awareness of how social engineers operate.

Conclusion

This paper analyzed the problems that current Web Vulnerability Scanners are facing when trying to detect XSS vulnerabilities, as reported in recent research it was found that the vulnerability scanners are a promising mechanism to fight the XSS vulnerabilities in web applications. One reason for the widespread of XSS vulnerabilities is that many developers aren't trained well enough. Current proposals allow to automatically identifying that kind of security holes, although they also present an important limitation: the accuracy of detecting can't satisfy the users need and the performance is low. In this work, a secure tool was developed which called PalXSS; this tool works in forum, takes input form field as a target to detect XSS attacks by injecting malicious JavaScript code.

Two factors were used to evaluate the new tool: the performance and accuracy. The average detection rate of PalXSS tool is 90.24% while the Acunetix is 57.31% and XSSploit is 24.39% in order. The results show the accuracy of PalXSS tool satisfying the users need than other tools. In addition, the execution time of the PalXSS tool had 137/sec, while the Acunetix and XSSploit had 147/sec,187/sec in order; this result shows that the performance of our tool have high performance and accuracy among other tools used in this work. The detection rate of PalXSS tool can satisfy the client's need, which gives the motivation to enhance the tool in the future work.

References

[1] Athanasopoulos, E. (2011). Modern Techniques for the Detection and Prevention of Web2. 0 Attacks (Doctoral dissertation, University of Crete)..

[2] B. Almurrani "Cross-Site-Scripting (XSS) Attacking and Defending" BACHELOR´S THESIS, ABSTRACT TURKU UNIVERSITY OF APPLIED SCIENCES Degree Program in Information Technology, Autumn 2009

[3] Cert advisory ca-2000-02 "malicious html tags embedded in client web requests. February 2000.

[4] Open Web Application Security Project. OWASP Web Application Scanner Specification Project. http://www.owasp.org/index.php/Category: OWASP Web Application Scanner Specification Project, 2010. [Online; retrieved June 19, 2010].

[5] Kirda, E., Kruegel, C., Vigna, G., & Jovanovic, N. (2006, April). Noxes: a client-side solution for mitigating cross-site scripting attacks. In Proceedings of the 2006 ACM symposium on Applied computing (pp. 330-337). ACM..

[6] Doupé, A., Cova, M., & Vigna, G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 111-131). Springer Berlin Heidelberg..

[7] Guido van Rossum Fred L. Drake, Jr., editor " Python Tutorial Release 2.3.3 "December 19, 2003.

[8] S. Christey and R. Martin, "Vulnerability type distributions in cve", version 1.1. [online], http://cwe.mitre. org/documents/vuln-trends/index.html, (09/11/07), May 2007.

[9] Hallaraker, O., & Vigna, G. (2005, June). Detecting malicious javascript code in mozilla. In Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on (pp. 85-94). IEEE..

[10] Wassermann, G., & Su, Z. (2008, May). Static detection of cross-site scripting vulnerabilities. In Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on (pp. 171-180). IEEE.

[11] Jovanovic, N., Kruegel, C., & Kirda, E. (2006, May). Pixy: A static analysis tool for detecting web application vulnerabilities. In Security and Privacy, 2006 IEEE Symposium on (pp. 6-pp). IEEE.

[12] Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y. (2004, May). Securing web application code by static analysis and runtime protection. In Proceedings of the 13th international conference on World Wide Web (pp. 40-52). ACM.

[13] Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G. (2007, February). Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In NDSS.

[14] Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G. (2008, May). Saner: Composing static and dynamic analysis to validate sanitization in web applications. In Security and Privacy, 2008. SP 2008. IEEE Symposium on (pp. 387-401). IEEE.

[15] P. Wurzinger, C. Platzer, C. Ludl, E. Kirda, and C. Kruegel. Swap: " Mitigating xss attacks using a reverse proxy". In Proceedings of the ICSE Workshop on Software Engineering for Secure Systems (SESS '09), 2009.

[16] C. Kruegel and G. Vigna. " Anomaly detection of web-based attacks". In Proceedings of the 10th ACM conference on Computer and communications security, pages 251–261. ACM New York, NY, USA, 2003.

[17] M. Johns, B. Engelmann, and J. Posegga. " Xssds: Serverside detection of cross-site scripting attacks". In Proceedings of the Annual Computer Security Applications Conference, pages 335–344. IEEE Computer Society Washington, DC, USA, 2008.

[18] E. Gal´an, A. Alcaide, A. Orfila, J. Blasco "A Multi–agent Scanner to Detect