An Analysis of the Effectiveness of Black-Box Web Application Scanners in Detection of Stored XSSI Vulnerabilities

Shafi Alassmi,Pavol Zavarsky, Dale Lindskog, Ron Ruhl, Ahmed Alasiri, Muteb Alzaidi

Master of Information Systems Security Management

Concordia University College of Alberta

Edmonton, Canada

alassmi.shafi@gmail.com, {pavol.zavarsky, dale.lindskog, ron.ruhl}@concordia.ab.ca, medo0o.2004@gmail.com, muteb.alzaidi@gmail.com

Abstract— **Stored Cross-Site Scripting (XSS) vulnerabilities are difficult to detect and state-of-the-art black-box scanners have low detection rates [1, 2]. Both Bau et al. and Doupe et al. investigated black-box web application security scanners, and this paper extends their analyses of state-of-the-art black-box detection of stored XSS. We use our own custom testbed, SimplifiedTB, which is available upon request. Weaknesses and limitations of black-box scanners identified in our study confirm weaknesses and limitations discussed by Bau et al. [1] and Doupé et al. [2]. The paper provides a list of recommendations for improving black-box detection of stored XSS vulnerabilities.**

Keywords—Stored Cross-Site Scripting Injection; XSSI vulnerabilities; black-box scanners

I. INTRODUCTION

Stored Cross-Site Scripting Injection (XSSI) is an attack that allows an attacker to bypass web application rules and policies. Such an attack takes a form of injecting malicious code into a storage area used by a web application; a common example would be a backend database. The malicious code is then executed on the user's browser when the web application retrieves data from storage to display it on the browser as page content. A very simple stored XSS example would be saving (i.e., storing) some JavaScript code, such as "<script>alert('Hello');</script>", on a forum post. Users who visit the post page will execute the JavaScript, and will be prompted with a popup window containing the message: "Hello". More sophisticated uses of stored XSS may result in an attacker gaining access to the victim's private information, such as the victim's "Session ID" to hijack his/her session and login as him/her. For example, the Samy worm that infected MySpace users started when the author injected JavaScript into his own profile page. When a victim visited Samy's page the injected script was executed, which both caused Samy to be added to the victim's friends list and injected the script into the victim's profile. As a result, when other users visited the victim's page they were infected with the same script as well [10]. Samy's script infected one million users in one day. This demonstrates how stored XSS can have a massive impact in a short time period.

Stored XSS may not be executed immediately; an attacker injects the code and waits for a victim to browse the page infected with the script. This adds to the challenges faced in detecting stored XSS vulnerabilities, especially by black-box scanners, as the reply from the web application server to be analyzed by scanners may not include the malicious script. For example, most web applications send a confirmation message of submission; therefore, if the injection was successful scanners will be required to revisit the page to confirm the injection, and in this way identify the existence of the stored XSS vulnerability.

The simplest exploitation of stored XSS vulnerabilities requires: (i) malicious code to be saved on the backend database of the web application and (ii) a user to browse the infected page that renders as part of its

content, yet without proper escaping or sanitization, any saved malicious code. . For example, suppose a user logs into a forum message board and posts a comment that includes the following script:

"<meta http-equiv="refresh" content=0; URL="http://www.evil.com/attack.html">."

When a victim user browses a page with the above post, the browser automatically redirects the user to the URL "www.evil.com/attack.html". Of course, the URL need not be an HTML web page; it might, e.g., be a link to a file that will be downloaded to the victim's machine. Some browsers (such as Internet Explorer) have introduced a feature called Cross-Site Scripting Filter. However, this feature is designed to help detect reflected XSS not stored XSS [8].

From 2007 to the present, all three types of XSS (reflected, stored, and DOM) have been on the top ten vulnerabilities list by the Open Web Application Security Project (OWASP) [3]. This paper presents an extension of the analyses of black-box scanners by Bau et al. [1] and Doupé et al. [2], but focusing exclusively on stored XSS detection. We employ a new testbed containing simpler XSS vulnerabilities, in order to better explain the weaknesses and limitations of state-of-the-art black-box scanners. We also list recommendations to enhance detection of stored XSS injections.

Our experimentation used the following three testbeds: PCI (also used by Bau et al. [1]), WackoPicko (also used by Doupé et al. [2]), and our SimplifiedTB. The first two testbeds were intentionally designed to represent complexities that exist in modern web applications, in order to test the practical effectiveness of black-box scanners' detection capabilities against known vulnerabilities. We developed the SimplifiedTB testbed to eliminate all complexities presented in the other testbeds. Our purpose in doing so was to render more transparent any weaknesses we found in these scanners' detection capabilities. Our overall research objectives are to: (i) test claims made by black-box web application security scanners in detecting stored XSS vulnerabilities, (ii) compare identified weaknesses and limitations with findings by Bau et al. [1] and Doupé et al. [2], (iii) reveal any progress made by black-box scanners in detecting stored XSS vulnerabilities, (iv) help black-box scanner users to understand the challenges faced when it comes to detecting stored XSS, and (v) contribute to best practices for detecting stored XSS with black-box scanners.

This paper is organized as follows. First, an example of a stored XSS injection vulnerability and its successful exploitation is described in Section II. Then, Section III reviews steps taken by black-box vulnerability scanners when scanning web applications; Section IV describes our research methodology; Section V discusses experimental results; Section VI provides recommendations; Section VII presents related work; and finally Section VIII concludes the paper and provides suggestions for future research.

II. STORED XSS INJECTION: AN ILLUSTRATION

Stored XSS vulnerabilities in web applications can be exploited in many ways. Every web page input entry that is not properly sanitized poses a threat. Doupé et al. [2] identified stored XSS vulnerabilities in the WackoPicko testbed. Moreover, they mentioned all un-sanitized entry points where a script can be injected and successfully stored in the backend database. In a real attack, an attacker would craft the script to be stored in the backend database. For example, a script might be crafted to steal cookies and, when executed, the collected cookies might be sent to a host under the attacker's control.

Successful exploitation of stored XSS vulnerabilities requires that an attacker understands the behaviors and functions of the web application. Vulnerabilities mentioned and discussed in the research of Doupé et al. [2] have been exploited in several ways; the attack being discussed in this section has not been previously identified as a stored XSS vulnerability. In WackoPicko testbed, when uploading a picture the "Tag" field is used as a directory name and the "File Name" field is set to be the name of the picture when constructing the URL that links directly to the picture. The problem is that the field name allows dots to be part of the name. It allowed us to create a simple script using a free source code editor and saved it as ".jpg" file format. When uploading the "picture" we used "MyFile.html" as the file name and "MyTag" as the picture tag. The script (stored XSS) was successfully uploaded and stored in the backend MySQL database of the web application. Due to the way WackoPicko constructs the URL of uploaded pictures, we can use the following URL: "http://wackopicko/upload/MyTag /MyFile.html" as a direct URL to file we uploaded. In a real scenario a link such as this is trusted by users as it directs users to the web application of their choice. When a user visits the page URL: "http://wackopicko/upload/MyTag /MyFile.html" the script stored in the "jpg" file is rendered out and the script embedded is executed by the browser.

III. BLACK-BOX WEB VULNERABILITY SCANNERS

Black-box scanners have strengths in detecting certain vulnerabilities. However, they have some limitations in detecting other vulnerabilities - such as detecting stored XSS vulnerabilities as it was shown by Beau et al. [1] and Doupé et al. [2]. For our experiments, we selected black-box scanners with claimed capabilities of detecting stored XSS vulnerabilities. The selected scanners are Acunetix Free Edition, N-Stalker Free Edition, Rational AppScan Enterprise, and Zed Attack Proxy (ZAP). Scanners used in our research are listed in Table 1.

TABLE 1 BLACK-BOX SCANNERS INFORMATION

Product	Vendor	Version	Edition
ZAP	OWASP	1.3.4	Free Open Source
N-Stalker WVS	N-Stalker	2012 (B 7.1.1.117)	Free
Acunetix WVS	Acunetix	7.0 (B 20111105)	Free
Rational AppScan	IBM	8.00.0.0 (B 444)	Enterprise

ZAP, N-Stalker, Acunetix, and Rational AppScan all go through similar scan phases when scanning a web application for vulnerabilities. The scanning phases performed by the scanners mentioned by Doupé et al. [2] and Khoury et al. [6] are verified to be same phases performed by versions of scanners listed in Table 1. The phases are the following:

1) Crawl: during this phase (a) the web application scanners capture relevant URLs and then (b) detect input fields within each page crawled; identify type of backend database; analyze error messages, etc. Pages with no input fields are ignored. However, dynamically created pages that may contain web user interface with potentially un-sanitized input fields are not included. When web application requires login, the scanners are configured to include login credentials prior to crawling.

2) Attack selection and penetration testing: scanners select predefined static attack vectors that match the previously identified entry points. Most scanners submit random data that is relevant to the selected XSS vulnerabilities scanning profile.

3) Response analysis: the response received contains what will be rendered to users on their browsers. Scanners receive the reply and analyze it to determine whether vulnerability is present or not. This phase is one of the challenges that scanners face as a scanner must make a decision based on comparing responses or subset of responses errors received with a predefined error list and then determines the vulnerability based on the error [6].

Section V discusses limitations and weaknesses discovered in the above three phases performed by black-box scanners when testing web application for stored XSS vulnerabilities.

IV. METHODOLGY

The performance of each scanner in detecting XSS vulnerabilities was analyzed against the three testbeds (i) PCI by Bau et al. [1], (ii) WackoPicko by Doupé et al. [2] available for download at the OWASP site [11], and (iii) against our own testbed named SimplifiedTB. We analyzed performance of scanners in the three phases (see Section III) including collected data and network traffic. Data collected includes: fields that have been identified, type of injections attempted, visited URLs, and type of data saved to backend database by black-box scanners.

A. Testbeds

The PCI testbed was designed and built by Bau et al. [1] at Stanford University to evaluate state-of-the-art scanners. The WackoPicko testbed was released for public use by Doupé el al. [2] from University of California, Santa Barbara. WackoPicko is now part of the OWASP Broken Web Applications Project, is available for free downloading and has well documented explanations of existing stored XSS vulnerabilities [11]. Both testbeds intentionally include stored XSS vulnerabilities with different challenges and complexities to measure the capability and performance of black-box scanners. PCI and WackoPicko testbeds contain stored XSS vulnerabilities that require user login and other stored XSS vulnerabilities that can be exploited without login anonymously. Both testbeds run on Linux, Apache, MySQL, and PHP (LAMP) systems.

To reduce the complexities in the two testbeds, SimplifiedTB was intentionally designed with simple-to-detect stored XSS vulnerabilities by black-box scanners. Our SimplifiedTB testbed was designed to contain three stored XSS vulnerabilities that do not require login. The testbed does not incorporate any filtering, escaping, encoding, or sanitization of inputs and all validations are disabled to make the web application vulnerable to stored XSS. It is developed based on ASP and MSSQL backend database. The source code of the testbed is available to interested readers upon request.

The black-box scanners used in this experiment were installed on a Windows (XP Professional) environment. We installed Wireshark on the same machine to monitor and capture the network traffic between the scanners and the deployed web applications.

TABLE 2 STORED XSS VULNERABILITIES IN TESTBEDS USED

Testbed	Total Stored XSS Vulnerabilities	Login Required	No Login Required
PCI	4	3	1
WackoPicko	3	2	1
SimplifiedTB	3	0	3

The PCI testbed was used and built by Bau et al. [1] to assess black-box scanners. Reviewing the source code revealed that PCI contains four stored XSS vulnerabilities in which three of them require user login.

WackoPicko testbed contains three stored XSS vulnerabilities in which only one requires login. Two of the vulnerabilities were previously identified in Doupé's et al. [2]. Through deeper analysis of the WackoPicko process of saving uploaded images we were able to upload script in the form of an image and then later execute it as illustrated in Section II. The identified stored XSS vulnerability related to the upload image field increases the total stored XSS vulnerabilities within WackoPicko to three vulnerabilities.

B. Black-Box Web Application Scanning Profiles

The black-box scanners used in this experiment have preset scanning profiles that we used to test the three testbeds for stored XSS vulnerabilities. Since two of the black-box scanners used in this experiment were limited to the use of XSS profiles, we set all scanners to only use XSS profiles. In addition we used the "Complete" scan profile in one of the scanners. The results of the comparison are discussed in Section V. We analyzed scanning performance using XSS profiles for all scanners.

C. Testing Steps

For each of the XSS scanning profiles we ran two scans. The first scan was without login credentials and the second included login credentials. For each scanner and to ensure we were getting results from the same testing environment we performed the scanning experiment by following the exact same steps as shown in Table 3. Table 3 lists all the steps taken prior to, during, and after scanning the testbeds. Prior to running the scan we ensured the testbed was clear from any transactions that may have affected the result of the scan. Such procedures were adapted from the research done by Khoury et al. [6]. During the scan we monitored the network traffic and recorded the traffic for further analysis. After completing scanning process, we gathered all collected data and analyzed it to understand the performance of the scanners in the detection of stored XSS vulnerabilities. Section V discusses the results of our analysis.

When the scanners failed to visit a page or to detect an input field in phase one (the crawling phase Section III) we manually crawled pages to ensure that scanners visited the page and identified the input fields to be tested when the scanning process started.

TABLE 3 EXPERIMENT PROCEDURES FOR EACH TESTBED

1. Restore web application to its initial state 2. Restore database to its original state 3. Configure scanners to use XSS profile with appropriate login information as needed 4. Set Wireshark to monitor the traffic between the scanners and the web server	**Prior to Scan**
5. Run Wireshark to monitor the network card that receives and sends traffic between scanner and web application 6. Start the black-box scanner	**During Scan**
7. Stop Wireshark and save relevant packets 8. Save scanning results 9. Export database entries for analysis	**Collecting Data**
10. Verify the detection rate 11. Analyze Wireshark files and database records 12. Repeat the previous steps with different scanner	**Analyzing Data**

V. RESULTS AND DISCUSSION

For every test we ran we monitored and recorded traffic between the scanner and the web application. After every test we exported and analyzed the records from the tested testbed backend database.

We ran two tests by each scanner for each testbed. One test included valid login credentials and the other ran without login credentials. Results from both tests for two scanners were almost identical and for the other two scanners were identical. There were not any major differences in the results collected in running both scans. For example, N-Stalker sent exactly 224 requests in both scans, with and without login, and results of both scans were identical. AppScan and Acunetix sent more requests when provided with login credentials, but the login credentials did not result in an increased detection rate and the stored XSS vulnerabilities detected were the same. For example, the multi-step stored XSS vulnerability in the WackoPicko testbed requires a valid login to enable scanners to access the page containing the vulnerability. The vulnerable page also introduces a major challenge to scanners since the first submission reviews the comment and another submission is required to store the input data into the database.

Moreover, we scanned all testbeds in a manual mode in which scanners record our activities on the web application and replicate them when running the scan. However, not all the performed activities in manual mode were successfully completed by the scanners. For example, we logged into the web application and uploaded a file and recorded the upload activity but none of the scanners were able to duplicate the upload activity successfully. By analyzing the backend MySQL database that records successfully completed activities, we confirmed that no scanner was able to upload images.

A. PCI Testbed – Results

Figure 1 compares scanners' performance by the number of visits per page and ability to create users in the database of PCI testbed when black-box scanners are scanning for XSS vulnerabilities.

Scanner 1 and scanner 2 were able to create more than 20 users; nevertheless, black-box scanners failed to use created user accounts to access pages available to the users.

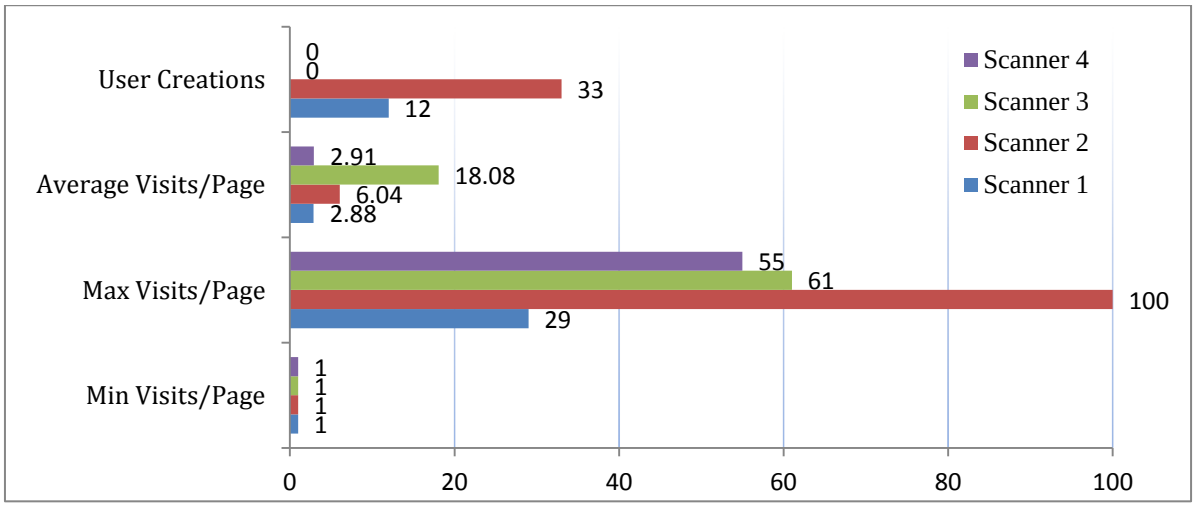

FIGURE 1 COMPARISON OF VISITED PAGES BY SCANNERS

Our analysis found that scanner 3 has a higher average in visiting pages of the web application. However, at the scan time the scanner 3 had the lowest detection rate of all scanners when it comes to stored XSS

vulnerabilities. This illustrates that the type of request and the type of attack vector are more important than the number of sent requests. The average detection rate of stored XSS vulnerabilities by all scanners came to 17.50%.

In Figure 1 reflects pages visited by scanners, not pages that have been missed. For example, "comment.php" was detected and was visited at least one time by all scanners.

Moreover, we ran scanners against the PCI testbed and results only included detection of stored XSS vulnerability that exists on the comment field on the "BookComments" page. The exploitation of the stored XSS vulnerability on "BookComments" page requires the following steps:

- Attacker navigates to the "BookComments" page
- Attacker types name, email, and comment field
- Attacker click "Submit" on the page to save it on the backend MySQL database

Although stored XSS vulnerability on "BookComments" is easy to detect only two scanners were able to detect the vulnerability and the other two failed to do so. Pages were login is required stored XSS vulnerabilities were not detected.

Figure 2 shows the detection rate of stored XSS for every scanner we used in the experiment. Black-box scanners detection rate of XSS stored vulnerabilities should not depend on the number of attacks executed or requests sent by the scanner against web applications as much as the variety of predefined attack vectors to be tested against. Scanners that had a higher number of requests sent to the server had a better chance of injecting more scripts into the server. However, many of the injected scripts were the same script injected into the same vulnerable field. Scanner 2 stored 16% more data in a form of repetition into WackoPicko database than scanner 1; however, scanner 1 detected the same number of vulnerabilities as scanner 2. The extra data that was stored by scanner 2 included random data and repetition which did not add more value to what was stored. For example, scanner 1 created 33 users in one of the testbeds in which 23 of them were identical user names and email addresses.

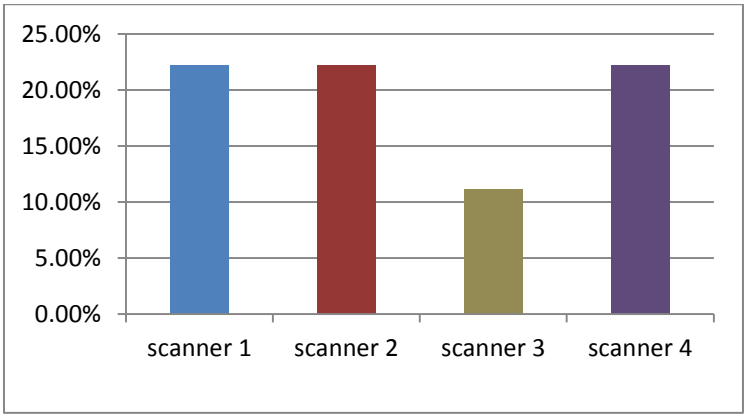

FIGURE 2 DETECTION RATE OF STORED XSS VULNERABILITIES

B. WackoPicko Testbed – Results

One of three stored XSS vulnerabilities in WackoPicko is a multi-step stored XSS vulnerability. A user posts a comment on an image then is presented with a preview screen before submitting the comment to the server. Steps required to successfully inject a malicious script are the following [2]:

- An attacker must be logged in to be able to select an image;
- An attacker selects an image to comment on;
- An attacker types a comment and submit it for the preview screen; and
- An attacker confirms the comment to be submitted

All scanners failed in detecting the multi-step stored XSS vulnerability because scanners lack an understanding of the interaction between both the "previewcomment.php" and "comments.php" pages. Scanners were not able to recognize that to be successful they must confirm the comment submission through the "previewcomment.php" page and therefore successfully store the malicious script. After analyzing the collected data we noticed that specific attack codes were performed against the preview comment page by all scanners. However, accessing the comment page directly has resulted in either not storing anything or storing nulls in the database. An example is that the script "<script>alert ('Hello');</script>" was entered on the comment field and submitted. The scanners did not realize that submission was just a redirect to a page to confirm the comment entered. Therefore, on the static confirmation page the comment field was left blank and submitted without the script above. This adds the challenges black-box scanners face when pages are created dynamically. Some scanners missed detecting dynamic pages and other scanners go in an infinite loop.

Black-box web application scanners did not perform well when interaction between a user and web applications was required as it is explained in the multi-step stored XSSvulnerability. No scanner was able to submit a comment successfully as the submit button redirects the user to a preview page before submitting the supplied data to the server. Such a simple task was not an easy task to be completed by black-box scanners which indicates that scanners need to be improved to understand interaction between pages within a web application.

Stored XSS injection does not necessary take an immediate action and our analysis of the network traffic revealed that scanners failed to revisit tested pages to confirm if rendering of injected script on browsers take place or not.

C. SimplifiedTB Testbed – Results

SimplifiedTB has two visible input fields and one hidden field. The hidden field requires a "Show" button to be clicked to make it visible. All three vulnerabilities can be easily exploited in two steps. An attacker will type in the malicious script and clicks "Submit". When the button is clicked the malicious script is saved into the database and only requires a refresh of the page to be able to see the script in action.

When we ran the four black-box scanners against SimplifiedTB, all scanners had the detection rate of 0%. No scanner was able to detect any stored XSS vulnerabilities existing in SimplifiedTB. All scanners successfully saved script in the database when tested against SimplifiedTB but failed to detect stored XSS vulnerabilities. The failure may be due to the fact that stored XSS injections impact is not immediate as we illustrated in Section I with the "Hello" message injection example and/or due to the analysis phase of the black-box scanners cycle. We strongly believe scanners failed to analyze correctly the response returned back from the server. Moreover, we believe that scanners failed to revisit the injected page to verify whether the injected script exploited the vulnerabilities or not. The approach used is more likely to work with reflected XSS injection since the impact is instant whereas stored XSS injection may require a second visit to the injected page to verify its impact and exploitation.

The detection of stored XSS vulnerabilities using the “XSS” profile completed the first two phases successfully but was poor in the third phase for all scanners. Scripts were successfully injected and stored in the backend database of our web application. However, the response returned by the web application was not analyzed properly which made all scanners miss detecting the vulnerability. To verify our findings we ran another scan with “Complete” profile to scan against every vulnerability category. The “Complete” profile resulted in a higher detection rate in comparison to “XSS” profile. Using “XSS” profile scanners, the scanners were able to crawl pages, identify entry points, inject attack vectors and successfully store them on the backend database; however, they were not able to analyze the response correctly to report the existence of the stored XSS vulnerabilities. We concluded that the issue is in the analysis phase where responses are analyzed. Black-box scanners should use enhanced approaches in response analysis depending on the profile selected.

During our analysis of running scanners with valid logins we found that scanners were able to complete all three phases of the scan. The first two phases were successful but the third phase failed to analyze responses correctly to detect the stored XSS vulnerabilities. Our findings were consistent with findings by Bau et al. [1] and Doupé et al. [2]. However, an issue that affects the performance of scanners is the fact that a same or similar attack in the penetration testing phase was being used for testing input fields against stored XSS vulnerability. An illustration, the only script that was used to test for stored XSS vulnerabilities was “<script>prompt (953552) </script>” . If black-box scanners used to track a script used to exploit stored XSS vulnerabilities fails, a different attack or a different representation of the attack should be used to enhance the chance of detecting stored XSS vulnerabilities. For example, if the script “<script>prompt (953552) </script>” fails to exploit stored XSS vulnerabilities scanners can try to launch a stored XSS injection using HTML tags such as “<iframe>”, rewrite the script using ASCII format and so on. Such an approach can enhance the detection rate of stored XSS vulnerabilities because it attempts to bypass input filtrations and tests against different injection formats.

Furthermore, black-box scanners test for stored XSS vulnerabilities by sending requests and analyzing the immediate response from the server. This affects the effectiveness of the scanners since stored XSS vulnerabilities are saved on the backend (i.e. database) of the web application to be executed at later time. Scanners should test for stored XSS vulnerabilities in a few steps starting from sending a legitimate request and then if an injection is successful - track what has been injected to revisit the page and validate if a stored XSS attack occur. Theoretically such an approach can be performed by:

- Scanners visit a pages and track scripts that exist
- Scanners injects malformed script and submits it to be saved in the backend
- Scanners revisits pages and analyzes the response to validate if injected scripts were successfully executed or can simply check if new script is introduced by comparing it to the track list of existing script

This can be done through analysis of scripts included in the original response and then flagging it as allowed and on the second response any added script should be flagged to be reviewed by analysts and testers.

In the above dicussed experiment we used two free edition black-box scanners that offered scanning against stored XSS vulnerabilities only. One of the scanners used was a full version which gave us the ability to scan testbeds using “XSS only” profile and “Complete” profile. “Complete” profile allows getting extra information for stored XSS vulnerabilities detection. Running the black-box scanner using the “Complete” profile showed

better performance in comparison to running the scanner under "XSS only" profile. We cannot assume that other scanners will have similar results since the results of the scanners were similar but not identical.

VI. RECOMMENDATIONS

Our recommendations are similar to Bau et al. [1] and Doupé et al. [2] recommendations because findings obtained from our experiment despite using newer version of the scanners were very similar to their findings. We recommend security professionals to configure black-box scanners with login credentials as Khoury et al. [6] have suggested and manually record activities that require interaction to enable some scanners to replicate the activities to enhance the detection rate of stored XSS vulnerabilities. Moreover, we recommend to end users to avoid the use of profiles that are specific to one vulnerability since the results showed that some stored XSS vulnerabilities were not detected using "XSS" profile but were detected using "Complete" profile.

Black-box scanners developers should focus on increasing coverage of predefined attack vectors and the different representation of the scripts to be injected. Scanners should try to use different representations of attack vectors in testing against stored XSS rather than increasing the number of tests with the same attack vector. For example, rather than using <script> tag to inject an alert script, scanners can try injecting an alert script in the format of ASCII or HEX-encoded values as some web applications may successfully filter the tag <script> but will fail to filter the ASCII representation of the tag or if it is combined with escape characters.

Black-box scanners help in detecting vulnerabilities; however, scanners do not point exactly to where the vulnerabilities are since scanners do not access the source code. Bau et al. [1] and Doupé et al. [2] emphasized that automated black-box scanners should be part of a security cycle rather than stand-alone security.

Scanners must be improved to understand the required interaction between pages such as confirmation and preview pages to enhance detection rate of stored XSS vulnerabilities. We recommend involving human testers as discussed by McAllister et al. [5] since human testers can understand the required interaction within a web application therefore, enhancing the likelihood of detecting stored XSS vulnerabilities.

Furthermore, when reviewing collected data, such as type of scripts injected by scanners, auditors can verify if it is necessary to try other attack vectors incase what is predefined by the scanners is not sufficient. Security professionals also need to realize that in some cases not all phases of the scanner cycle are completely performed. As it was illustrated in testing example against the SimplifiedTB, scripts were injected successfully but were not detected as vulnerabilities in the analysis phase.

VII. RELATED WORK

Many researchers have shown commitment in their researches to help explain limitations on web application security and recommended ways to improve upon the security of web applications. Many black-box scanners have been tested and limitations of such scanners have been discovered and discussed in details.

Bau et al. [1] showed that state-of-the-art black-box scanners have a low detection rate when it comes to stored XSS. It was recommended to use black-box scanners as part of a security program rather than a stand-alone system due to the fact of difficulties in designing tests against stored XSS to detect the vulnerabilities. Zang et al. [4] presented what they called "Execution-flow" approach to help detecting stored XSS. The approach introduced is named "Runtime Monitoring Process".

This approach is achieved by matching all calls made to JavaScript functions using FireFox Dtrace to a trusted list of JavaScript functions. If a malicious code is detected then it is removed and the page sent to the end user browser without the injected script. The Doupé et al. [2] paper confirmed that all tested eleven scanners have missed detecting stored XSS which requires a confirmation step. The Doupé et al. [2] WackoPicko testbed included some of the complexities implemented in modern web applications. Some of the limitation discovered by Doupé et al. [2] was that each request by the scanners handles one parameter only. This approach can impact the effectiveness of the scanners since the ignored field may be required and some scanners leave it blank. McAllister et al. [5] mentions that scanners fail to submit comments when the first submission only shows a preview of the comment to the user before the final submission of the comment. Therefore, McAllister et al. [5] claims that their tool was able to detect stored XSS by generating enough requests to reach every entry point rather than only the ability to inject a malicious input. McAllister et al. [5] also claims their approach can be applied to other injection types such as SQLI; however, Khoury et al. [6] mentions that "we cannot assume that the tool proposed by McAllister et al. [5] can detect stored SQL injections". Such a disagreement leaves the relationship between XSS and SQL stored injections vague and in need of researchers to conduct experiments to confirm if detecting stored XSS vulnerability can enhance the detection rate of stored SQL - or vice versa.

VIII. CONCLUSION

In our analysis we extended the analysis completed by Bau et al. [1] and Doupé et al [2] by creating our own custom testbed SimplifiedTB. Unlike the first two testbeds, SimplifiedTB was designed as simple as possible to eliminate all complexities such as authentication and login. Our performance analysis was based on testing four black-box web vulnerabilities scanners against the three testbed. Our results showed that the first two phases performed by the black-box scanners were successful; however, the third phase was not. The detection rate of stored XSS vulnerabilities using state-of-the-art black-box scanners is low and needs to be improved. We verified major challenges that face black-box scanners were: type of predefined attack vector selected to test against stored XSS, lack of understanding of required interaction between pages, and weak analysis of responses returned by the server. Although we used "XSS" profile to scan against stored XSS vulnerabilities scanners failed to detect the vulnerabilities. Scanners performed better in detecting stored XSS when a "Complete" profile was selected. We strongly recommend: end users perform a full scan to enhance results received from black-box scanners, and security professionals integrate automated black-box scanners into a security cycle rather than rely on it as a stand-alone protection approach.

ACKNOWLEDGMENTS

We would like to thank Bau et al. [1] from Stanford University and Khoury et al. [6] for providing the testbed PCI we used to conduct our experiments. Thanks also go to Acunetix, IBM, ZAP and N-Stalker for participating in this research.

REFERENCES

[1] Jason Bau, Elie Bursztein, Divij Gupta, John Mitchell, "State of the Art: Automated black-box Web Application Vulnerability Testing" May 2010.

[2] Adam Doupé, Marco Cova, and Giovanni Vigna, "Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners", July 2010.

[3] Open Web Application Security Project. (2010). [Online]. Available: http://www.owasp.org/index.php /Category:OWASP_Top_Ten_Project

[4] Qianjie Zhang, Hao Chen, Jianhua Sun, "An Execution-flow Based Method for Detecting Cross-Site Scripting Attacks" June 2010.
[5] Sean McAllister, Engin Kirda, and Christopher Kruegel, "Leveraging User Interactions for In-Depth Testing of Web Applications", 2008.
[6] Nidal Khoury, Pavol Zavarsky, Dale Lindskog, and Ron Ruhl, "An Analysis of black-box Web Application Security Scanners against Stored SQL Injection", 2010
[7] CENZIC Enterprise Application Security (2012) http://www.cenzic.com/downloads/Whitebox_VS_Blackbox_WP.pdf
[8] Michael Brooks, "Bypassing Internet Explorer's XSS Filter", 2011
[9] John B. Dickson, "Black Box versus White-Box: Different App Testing Strategies"
[10] Jeremiah Grossman, "Cross-Site Scripting Worms & Viruses: The Impending Threat & the Best Defense", June 2007.
[11] OWASP Broken Web Applications Virtual Machine. http://code.google.com/p/owaspbwa/wiki/Downloads