Ss. Cyril and Methodius University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

2014

Proceedings of the 11th International Conference for Informatics and Information Technology

Held at Hotel Molika, Bitola, Macedonia
11-13th April, 2014

Editors:
Vangel V. Ajanovski
Gjorgji Madjarov

ISBN 978-608-4699-04-0

One Unwanted Feature of Many Web Vulnerability Scanners

Nataša Šuteva, Dragan Anastasov, Aleksandra Mileva
Faculty of Computer Science, UGD
Štip, Macedonia
{natasa.suteva, dragan.anastasov, aleksandra.mileva}@ugd.edu.mk

***Abstract* - Security experts, web developers, hackers sometimes use Web Vulnerability Scanners (WVSs) for identifying vulnerabilities in web applications. There are commercial and free/open source WVSs, and nowadays, many companies offer WVSs as services. In this paper, we test and evaluate 3 free/open source WVSs and 4 free, trial or regular editions of commercial WVSs using two versions of our one created trading web application. One version has SQL Injection and XSS vulnerabilities as critical, and the other version is free from these vulnerabilities. Results are showing that most of the scanners pollute the backend database with many garbage records using user input fields for obtaining user's opinion, comments, rating, etc., independently of the presence or absence of given critical vulnerabilities. In our experiment, garbage records were injected as comments for ads, and the magnitude of pollution goes more than 50 times the number of ads in the database in the worst case. Also, some scanners manage to find the implemented vulnerabilities without producing garbage records.**

Keywords—Web Vulnerability Scanners, backend database, garbage records

I. INTRODUCTION

Web Application Security Scanners (WASSs) or Web Vulnerability Scanners (WVSs) are a type of security software, most commonly used by website owners, security experts and hackers, to perform identification of potential vulnerabilities in the web applications, independent of the particular technology used for their implementation. They access the web applications in the same manner as user does, through the web front-end. Usually they are black-box testers, because they do not have access to the source code. Vulnerability detection mechanisms and scans differ in different WVSs, from looking at registry entries in MS Windows operating systems to see if a specific patch or update has been implemented, modifying URLs to check for sanitization issues or discover known vulnerabilities, to actually performing attacks on detecting vulnerabilities. The OWASP (Open Web Application Security Project) Top Ten 2013 vulnerability list [14] is often used as a minimum standard for website vulnerability assessment and PCI compliance according to the Payment Card Industry Data Security Standard (PCI DSS) [9], so performing web vulnerability scans is a necessity for PCI compliance. Additionally, the usefulness of WVSs comes from automatically and cost-effective conduction of security checks and production of the final report, which often includes a remedy for found vulnerability.

On the other side, WVSs are not a silver bullet, capable of detecting all of the possible vulnerabilities and attack vectors that exist. There are several reports showing that today WVSs fail to detect a significant number of vulnerabilities in test applications [2, 4, 5, 7, 8, 10, 11, 12, 13, 15].

Another big issue about WVSs is can they harm in any way tested web sites? Black box scanners have tendency to perform invasive scans, which sometimes can cause email floods, as well as publishing of garbage blog posts, garbage comments, ratings, etc [1]. Grossman [6] shares their experiences from ten years of scanning tens of thousands of real-live websites of all shapes and sizes. He gives the following 7 ways how some WVSs can harm scanned web site:

- Following "Sensitive" Hyperlinks – some web sites have hyperlinks (GET requests) that, when clicked, execute backend functionality that deletes data, cancel orders, remits payment, removes user accounts, disables functionality, and etc.
- Automatically Testing "Sensitive" Web Forms – sometimes submission of a Web form (POST request) may generate emails to customer support, execute computationally expensive backend processes, direct submitted data that will be visible to other users, and so on. This can result in spamming inboxes with thousands of emails, taking down the website due to resource load, negatively impacting the user experience of the entire user-base by showing them unexpected data, and costing the company large sums of money
- Poorly Designed Vulnerability Tests – during dynamically testing, various meta-character strings are put into input fields, URLs, POST bodies, headers, etc. Website can be harmed when it mistakes meta-characters for executable code.
- Connection Denial of Service (DoS) – sometimes scanning requires sending hundreds of requests simultaneously to the website, so this can easily exhaust a website's available connection pool and render the system unable to serve legitimate visitors.

- Session Exhaustion DoS – complete testing a website requires that vulnerability scans are run in an authenticated state. When WVS logs in hundreds of times during testing, it may consume all the website's session credential resources, and no additional legitimate users can log-in, until the session credential garbage collection is conducted.
- CPU DoS – some websites have computationally expensive hyperlinks, which during the scans may be clicked a large number of times, contrary to what was expected, and consume all of a websites available CPU resources.
- Verbose Logging and Run-Time Error – scanning can involve a large number of abnormal requests, which could raise various backend application exceptions and verbose run-time error logging. Because of this, the disk size of the logs generated and stored could be substantial.

Consequently, the vulnerability scans need to be performed with precautions, and, ideally, a replica of the live environment should be created in a test lab, so if something goes wrong, only the replica is affected. At least, before starting scans, latest backups are needed. Some automated scanners include settings for launching a non-invasive scans, but these kind of scans will only launch some very basic "security" checks against the target, such as text searches, file checks, version checks and some other basic tests, which typically do not lead to a malicious defacement of the site or web application. So, invasive scans are necessary, because if an automated WVS can break down tested website, a malicious user can do even worse.

In this paper, we try to measure the amount of generated garbage records per scan, by testing 3 free/open source WVSs and 4 free, trial or regular editions of commercial WVSs, with consideration of scanner's capability to detect several basic critical/important vulnerabilities. We want to see is it possible to detect these vulnerabilities, with performing non-invasive scans, in the sense that scanners do not leave any garbage records. Also, it was interesting to see if the pollution of database obtained by scanning, depends on the presence or absence of these vulnerabilities in the web application. After Introduction, Section II gives the basic architecture of the black box WVSs. In Section III we give a brief explanation of two versions of used testbed web application and seven WVSs with their general characteristics and input vector support, followed by used methodology, obtained results on the measured number of garbage records, and discussion. At the end, we give short concluding remarks.

II. BLACK BOX WEB VULNERABILITY SCANNERS

Generally, the core of the WVSs is made up from three main components: a crawling component, an attacker component and an analysis component.

First, the user enters at least one URL, with or without user credentials for the given web application, and then the crawling component identifies all the reachable pages in the application, and all the input points to the application. After the user sets the scanning profile, the scanner can proceed automatically or by user interaction. We used only automated mode for our experiments.

Once the crawling component finishes its job, the next components perform analysis of the discovered data, and for each web form, for each input and for each vulnerability type for which the WVS has test vectors, the component generates values that are likely to trigger a vulnerability. Then, the form content is sent to the web server as an HTTP request, and after processing the request, the server sends back a response via HTTP.

The attacker component analyzes discovered data and for each web form, for each input and for each vulnerability type for which the WVS has test vectors, the attacker module generates values that are likely to trigger a vulnerability. Then, the form content is sent to the web server using either a GET or POST request, and appropriate response is obtained from the server via HTTP.

Finally, the analysis component performs parsing and interpreting the server response. Decision if a given attack was successful is made by calculation of confidence value, by implementing attack-specific response criteria and keywords.

III. EXPERIMENTS AND RESULTS

A. Testbed Web Application

We created a simple trading web application, where unregistered users can list ads, see information and description about individual ad, comment on the ad and so on. Registered users can add ads and manage ads. We created two versions of the application, a vulnerable and a safe one. The vulnerable version is affected by SQL injection (in 3 scripts), reflected and stored XSS vulnerabilities.

The web server hosting our web applications run on 64-bit Windows 8.1 Enterprise operating system. The following technologies are used: Apache server version 2.4.4, PHP version 5.4.12 and MySQL version 5.6.12.

B. Tested Web Vulnerabilities Scanners

The scanners were run on a machine with an Intel (R) Core (TM) i7-3632QM 2 x 2.20GHz CPU, 6 GB of RAM, and 64-bit Windows 8.

Table 1 lists the seven WVSs used in our study and their general characteristics. All have a graphical user interface and support for proxy mode (manual crawling). Three of them, NetSparker Community Edition, N-Stalker X Free Edition and Acunetix WVS run only on Windows, and other four can be installed on Linux and OS X also. Only N-Stalker X Free Edition, OWASP ZAP and IBM Rational AppScan can produce a report. Their input vector support is given in Table 2. Many different characteristic comparisons with older versions of these WVSs can be found on Chen's web site SecToolMarket [3].

Free NetSparker Community Edition has many features disabled, compared to its commercial version, but still you can scan and exploit SQL injection and XSS vulnerabilities without any false-positives.

TABLE 1: GENERAL CHARACTERISTICS OF THE EVALUATED SCANNERS

	NetSparker Community Edition	N-Stalker X Free Edition	OWASP ZAP	IronWASP	Vega	Acunetix WVS	IBM Rational AppScan
Company/ Creator	Mavituna Security	N-Stalker	OWASP	L. Kuppan	Sub-graph	Acunetix	IBM
Version	3.1	X-build	2.2.2	2013 beta	1.0	9	7.8
Released			Sep. 2013				
Licence/ Technology	Freeware .Net 3.5	Freeware Unknown	ASF2 Java 1.6.x	GNU .Net 2.0	EPL1 Java 1.6.x	Trial AcuSensor	Comm. Unknown
Operating System	Windows	Windows	Windows Linux OS X	Windows Linux OS X	Windows Linux OS X	Windows	Windows Linux OS X
Report	No	Yes	Yes	No	No	No	Yes
Scan Log	Yes	No	Yes	Yes	Yes	Yes	Yes

N-Stalker X Free Edition provides a restricted set of features, compared to its commercial version, and will inspect up to 100 pages within the target application. It offers a restricted version of the N - Stealth Database, web server security check, reduced analysis of web signature attacks, etc.

TABLE 2: SUPPORTING INPUT VECTORS BY THE EVALUATED SCANNERS

	NetSparker Community Edition	N-Stalker X Free Edition	OWASP ZAP	IronWASP	Vega	Acunetix WVS	IBM Rational AppScan
HTTP Query String Parameters	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HTTP Body Parameters	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HTTP Cookie Parameters	Yes	Yes		Yes		Yes	Yes
HTTP Headers	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HTTP Parameter Names				Yes			Yes
XML Element Content	Yes		Yes	Yes		Yes	Yes
XML Attributes	Yes		Yes	Yes		Yes	Yes
XML Tags							
JSON Parameters	Yes		Yes	Yes		Yes	Yes
Flash Action Message Format							Yes
Custom Input Vector				Yes			Yes
SUMMARY	**7**	**4**	**6**	**9**	**3**	**7**	**10**

OWASP Zed Attack Proxy (ZAP) is a free and open source, easy to use, integrated scanning and penetration testing tool, and it is designed to be used by people with a wide range of security experience. ZAP includes intercepting proxy, active and passive scanners, traditional and Ajax spiders, WebSocket support, fuzzing, forced browsing, port scanner, script console, etc.

IronWASP (Iron Web application Advanced Security testing Platform) is a free and open source tool, created by Lavakumar Kuppan. It offers full and semi-automated scans, JavaScript static analysis, scripting shell for Python and Ruby giving full access to the IronWASP framework, and this can be used by the pen testers to write their own fuzzers, create custom crafted request, analysis of logs, etc. Another its strength is the possibility of using different external libraries like IronPython, IronRuby, FiddleCore, etc.

Vega is a free and open source automated scanner for quick tests and an intercepting proxy for tactical inspection.

For this test we are using fully functional 14-day trial version of Acunetix WVS. This scanner uses AcuSensor Technology, and besides scanning, it offers advanced penetration testing tools.

IBM Rational AppScan, now known as IBM Security AppScan, is a family of web security testing and monitoring tools from the IBM. For our tests, we used older version of IBM Rational AppScan.

C. Methodology

In our experiments, scanners were run without logging, and only the default values for configuration parameters were used. Only N-Stalker X Free Edition was run with OWASP policy.

Backend database consists of 3 tables, with initially 3 users, 7 ads, and no comments. After every scanning we summed the number of garbage comments in the database generated by the scanner and the number of affected ads, and

by deleting the comments, we prepare the database for the next scan. For every scanner we made 3 scans on the web applications.

D. Results and discussion

Table 3 shows the capabilities of tested WVSs for finding critical/important vulnerabilities. We need this to see how leaving garbage comments is connected with this capability. Only N-Stalker X Free Edition cannot find SQL vulnerability, and OWASP ZAP cannot find reflected XSS. (2/3) means that the scanner had identified only two of three vulnerable scripts.

TABLE 3: FOUNDED CRITICAL/IMPORTANT VULNERABILITIES

	SQLI	Reflected XSS	Stored XSS
NetSparker Community Edition	Yes (3/3)	Yes	Yes
N-Stalker X Free Edition		Yes	Yes
OWASP ZAP	Yes (2/3)		Yes
IronWASP	Yes (2/3)	Yes	Yes
Vega	Yes (3/3)	Yes	Yes
Acunetix WVS	Yes (2/3)	Yes	Yes
IBM Rational AppScan	Yes (2/3)	Yes	Yes

Table 4 and Table 5 give the number of garbage comments produced by the tested scanners in 3 independent scans on the safe and the vulnerable test application, respectfully.

TABLE 4: NUMBER OF GARBAGE COMMENTS FOR THE SAFE TESTBED WEB APPLICATION FOR 3 SCANS

	Number of garbage comments			Ads
	Scan 1	Scan2	Scan 3	
NetSparker Community Edition	156	160	156	All
N-Stalker X Free Edition	26	26	26	All
OWASP ZAP	61	61	61	All
IronWASP	0	0	0	-
Vega	0	0	0	-
Acunetix WVS	367	367	367	All
IBM Rational AppScan	52	52	51	All

One can see, that these numbers, ranges from 0 to 367 for the safe web application and from 0 to 180 for the vulnerable web application, and that for all scanners that produce garbage comments, all ads are affected. This means that if our database have thousands or more adds, which is the situation in reality, one scan with these scanners will produce at least the same number of the garbage comments. Two scanners, IronWASP and Vega, do not leave any garbage comments, but are capable of finding given vulnerabilities (IronWASP find 2 of 3 vulnerable scripts for SQLI). These results mean that some WVS can find tested critical/important vulnerability, without necessity to use invasive techniques. Nothing can be concluded about finding other vulnerabilities without invasive scans.

TABLE 5: NUMBER OF GARBAGE COMMENTS FOR THE VULNERABLE TESTBED WEB APPLICATION FOR 3 SCANS

	Number of garbage comments			Ads
	Scan 1	Scan 2	Scan 3	
NetSparker Community Edition	156	150	156	All
N-Stalker X Free Edition	10	10	10	All
OWASP ZAP	210	210	210	All
Iron WASP	0	0	0	-
Vega	0	0	0	-
Acunetix WVS	144	144	144	All
IBM Rational AppScan	178	178	180	All

Acunetix WVS leaves most garbage comments for the safe web application, with a magnitude of more than 50 times larger than the number of ads in the tested database. OWASP ZAP leaves most garbage comments for the vulnerable web application - 30 times larger than the number of ads in the tested database.

The experiments also show that WVSs that create garbage records, do that even when web application is free from critical/important vulnerabilities. Some WVSs, like N-Stalker X Free Edition, OWASP ZAP and Acunetix WVS produce more garbage comments for the safe web application, while IBM Rational AppScan produces more garbage comments for the vulnerable web application. First behavior is easier to understand, and can be explained that WVS stop testing the script on giving vulnerability, after it found it.

Also, some scanners, like NetSparker Community Edition and IBM Rational AppScan produce different numbers of garbage comments, but with small deviation, for scanning the same web application.

IV. CONCLUSIONS

Our experiments show that different scanners produce different numbers of garbage records in the backend database, and because of that, when we use them for scanning, web administrators need to make a backup of their database. This can protect them from spending additional time after scanning, for cleaning the database. Also, our experiments show that some scanners have capabilities of finding tested critical/important vulnerabilities, without using invasive techniques that produce garbage records. WVSs that produce garbage records, do that regardless of presence of a given vulnerability in the web application.

REFERENCES

[1] R. Abela, "A complete guide to securing a website", Acunetix [Online]. Available: http://www.acunetix.com/websitesecurity/website-auditing-wp/

[2] J. Bau, E. Bursztein, D. Gupta and J. Mitchell, "State of the art: automated black-box web application vulnerability testing", In Proceedings of the IEEE Symposium on Security and Privacy, May 2010.

[3] S. Chen, SecToolMarket, [Online]. Available: http://sectoolmarket.com/

[4] A. Doupe, M. Cova and G. Vigna, "Why Johnny can't pentest: an analysis of black-box web vulnerability scanners". In C. Kreibich, M. Jahne (Eds.) Proceedings of the 7th International conference on Detection of Intrusions and Malware, and Vulnerability Assessment - DIMVA'10, pp. 111-131, Springer Berlin Heidelberg 2010.

[5] J. Fonseca, M. Vieira, and H. Madeira, "Testing and comparing web vulnerability scanning tools for sql injection and xss attacks", In Proceedings of the 13th IEEE Pacific Rim International Symposium. Dependable Computing (PRDC 2007), vol. 0, 2007, pp. 365–372.

[6] J. Grossman, "7 ways vulnerability scanners may harm website(s) and what to do about it", WhiteHat Security 2012, [Online]. Available: http://blog.whitehatsec.com/7-ways-vulnerability-scanners-may-harm-websites-and-what-to-do-about-it/

[7] N. Khoury, P. Zavarsky, D. Lindskog and R. Ruhl, "Testing and assessing web vulnerability scanners for persistent SQL injection attacks", First International Workshop on Security and Privacy Preserving in e-Societies (SeceS '11), New York, NY, USA, 2011.

[8] N. Khoury, P. Zavarsky, D. Lindskog and R. Ruhl, "An analysis of black-box web application security scanners against stored SQL Injection", In Proceedings of the IEEE Third International Conference on Privacy, Security, Risk and Trust (PASSAT 2011) and 2011 IEEE Third International Conference on Social Computing (SOCIALCOM 2011), Boston, USA, October 2011.

[9] Payment Card Industry Security Standards Council. (PCI) Data Security Standard: Requirements and Security Assessment Procedures. October 2010. [Online]. Available: https:// www.pcisecuritystandards.org/documents/pci_dss_v2.pdf.

[10] H. Peine, " Security test tools for web applications". Technical Report 048.06, Fraunhofer IESE (January 2006)

[11] L. Suto, "Analyzing the effectiveness and coverage of web application security scanners", [Online]. October 2007. Available: http://www.stratdat.com/webscan.pdf.

[12] L. Suto, "Analyzing the accuracy and time costs of web application security scanners", [Online]. Feb 2010. Available: http://ha.ckers.org/files/Accuracy and Time Costs of Web App Scanners.pdf

[13] N. Šuteva, D. Zlatkovski, A. Mileva, "Evaluation and testing of several free/open source web vulnerability scanners", In Proceedings of the 10th International conference on Informatics and Information Technology (CIIT 2013), 2013, pp. 221-224.

[14] Open Web Application Security Project, "OWASP Top Ten Project" [Online]. Available: http://www.owasp.org/index.php/Category: OWASP Top Ten Project.

[15] A. Wiegenstein, F. Weidemann, M. Schumacher, S. Schinzel, "Web Application Vulnerability Scanners—a Benchmark". Technical Report, Virtual Forge GmbH (October 2006)