Identification of Performance Issues in Contemporary Black-Box Web Application Scanners in SQLI

HA THANH LE and PETER KOK KEONG LOH
Computer Security Laboratory
Nanyang Technological University, Singapore
lehathanh@ntu.edu.sg

Abstract: - We evaluate the performance of seven commercial and open-source black-box scanners in scanning for server-side vulnerabilities, particularly for SQL injection. We identify performance issues in black-box Web application scanning. We describe our experiments on our test-bed environment. The results are then analyzed and revealed important performance issues limiting the scanning capability of the selected scanners. Based on these findings, we give suggestions to improve SQLI black-box scanning with improved site discovery as well as active, context-aware SQLI vulnerability detection.

Key-Words: - Web application scanner, black box scan, SQLI vulnerability scan, active context-aware vulnerability detection.

1 Introduction

Our research work may partially overlap with other existing research (such as [1, 2]). However, in our work, we focus on a different set of characteristics and measurements used to evaluate and determine the strength, weakness and gaps that exist in selected black-box scanners. This approach reveals new performance issues and suggests new approaches in SQLI detection research.

The paper is organized as follows. In section 2, we review recent detection techniques in SQLI black-box scanning. We describe the scope and methodology of our work in section 3. Next, in section 4, we describe our test environment and how we conduct the tests. Results and analysis are detailed in section 5. In this section, we also present findings from evaluation and proposal for Black box Web scanning research that need to focus in the next stage. Section 6 concludes this paper followed by the references.

2 Literature Survey

2.1 SQLI Detection Techniques

SQL injection is a technique for exploiting Web applications with malicious client-supplied inputs. A characteristic diagnostic feature of SQL injection attacks is that they change the intended structure of queries issued. When testing for SQLI, we need to specify and include the various injection types such as direct injection, quoted injection, SQL command with system commands injected, parameter-based injection via Web-based forms, and error-prone injection [3].

There are many SQLI detection techniques. For example, the parsing technique is used to detect SQL command injection attacks (Su and Wasseman [4]). Tainting technique [5] is used to detect vulnerable location and data. Most of these are based on traditional signature detection techniques. They inspect Web traffic to identify SQL injection-related request/responses [6].

Das, Sharma, and Bhattacharyya [7] pointed out that the main issue in most SQLI attack detection approaches is developer-dependent. Either trusted input strings or unsafe input characters are initially generated and validated by developer. Hence, the reliability of proposed techniques rely heavily on developer’s skill. A new detection technique (DUD) for SQL injection based on dynamic adaptive query matching was proposed in order to overcome this drawback. However, the *edit distance* threshold which is the key parameter of automatic detection of injected queries is a user-dependent heuristic. Detection efficiency will rely on a tester’s skill in selecting an optimum edit distance.

An attacker can even exploit SQLI vulnerabilities to determine if injected SQL statements execute even when error messages and query results are not returned to the client’s browser [8]. The injection strings, in such cases, can be customized to the specific database system (SQL Server) and target system (IP address and port). The detection technique could then be based on selective fuzzing with parameterized queries or non-alphanumerical characters as user inputs.

A prototype SQL injection detection system (SQL-IDS) reported in [9] employs a detection technique that utilizes specifications which define the intended syntactic structure of SQL queries that are produced and executed by the Web application. The application is then monitored for executing queries that are in violation of the specification.

Sunitha and Sridevi [10] proposed a detection technique that performs syntax-aware evaluation of a query string immediately before the string is sent to the database to be executed. Bisht et al. [11] proposed a mining technique for detecting SQL injection. Their approach is to dynamically mine the programmer-intended query structure, and detect attacks by comparing against the structure of the actual query issued. The system, *Caddid*, may be trained with runs over benign candidate inputs.

A recent technique presented in [12] is effective only for SQL injections that insert a tautology in the SQL queries, but cannot detect other types of SQL injection attacks. Another technique uses available cheat sheets (such as [13-15]) for testing the Web applications. Although this is not an adaptive solution but if the tester/attacker has done thorough analysis of the target application, customized injection cheat sheet would be very effective.

In next section, we will describe our work in testing black-box web application scanners.

3 Scope and Methodology

In our investigations, we evaluate the scanning capability of selected black-box Web application scanners. The evaluation includes:

- The overall functionality of the scanners (a.k.a. vendor's product specification) (T1.1).
- The capability in providing additional functions that support finding specific vulnerabilities (T1.2).
- The capability of detecting known existing vulnerabilities in the Web application: the types of vulnerability that can be detected from a specific Web application (T2).
- The scanning coverage. Here, we try to discover the scanners' vulnerability database from which a scanner can perform automatic scanning (T3.1). We also need other evaluation parameters on runtime scanning to determine the actual scanning coverage.
- The positive detection ratio versus negative detection ratio with the aid of manual scans (T3.2).
- The specific tests that a scanner performs on a Web application in order to detect vulnerability (T4).
- The import-export capability: a scanner may perform better if it is able to import pre-scanning settings from and/or export post-scanning results to other analysis tools (T5).

Default settings are used for all scanners to remove the dependency on testing expertise. For example, when scanners are selectively combined to get more coverage or additional tools used to strengthen scanning results (an external proxy tool and/or a search engine's database may be used to establish a thorough map and UI of a Web application during pre-scanning stage).

In next section, we present our test environment set-up for the experiments.

4 Experimental Set-up

4.1 Platform

The test environment is illustrated in Fig. 1.

- In-house environment: comprised of a LAN of 4 computers:
  - Server: We deployed virtual machine-based servers on the available hardware. Installed server platforms include:
    - Windows XP SP3 with WAMP (Windows-based Apache, MySQL, and PHP)
    - Windows 2003 Server/Windows 2008 Server
    - Linux Server (Ubuntu) in PC1
    - A dedicated Solaris machine and other OS images (e.g. FreeBSD, Vyatta firewall OS) are installed for later usage.
  - Security prevention is disabled to minimize false negatives in the experiments. Specifically, we do not use IDS, IPS, filters or honeypot systems. Firewalls (mostly personal OS-integrated firewall) are disabled. In case a firewall is enabled, default settings are used with considerations for test applications and scanners, network connection, protocol and opened ports.

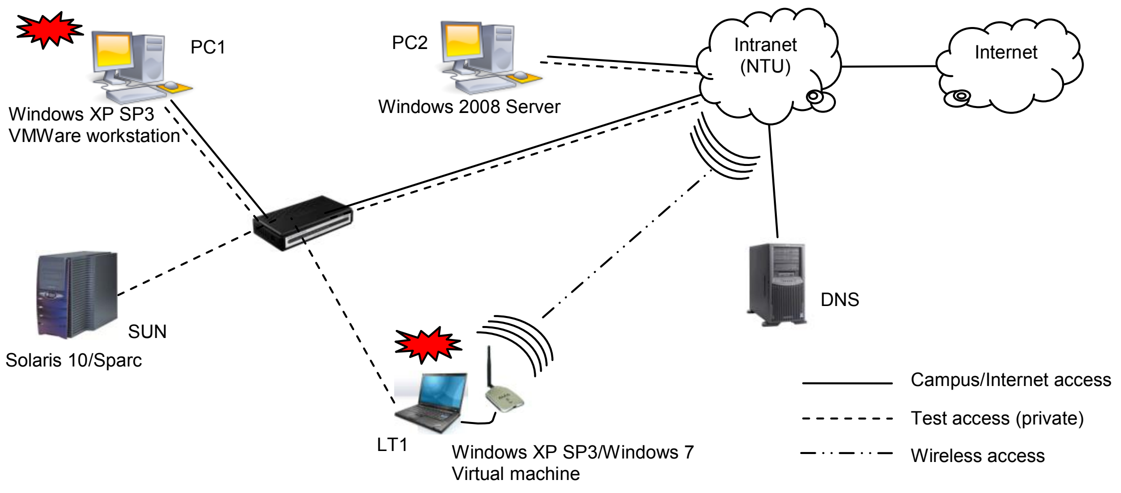

Fig. 1 Test Environment

- Client: the scanners are installed in machines that play client-role/attack role. Client OSs include:
  - Windows XP SP3
  - Windows 7 Ultimate (evaluation)
  - Images of Ubuntu-based BackTrack, Samurai Web Testing Framework, BackBox or Fedora-based system are used interchangeably depending on the required open source scanners / tools.
- DNS: a DNS server is used when a domain name is assigned to the deployed Web application (or Website). In this case, the servers at PC1 and PC2 will be named accordingly without altering the intranet configuration. Scanning is performed either with domain names or IP addresses.

- Online environment: vulnerability-ready Web applications are deployed as test-cases. Hence, we can perform scanning flexibly against these applications without violating and raising sensitive security considerations. For these applications, all we need is a computer with internet access and scanners installed.

4.2 Target Applications

After setting up the test environment, we installed the scanners and performed scanning on the pre-configured Web applications. For each test, only specified applications were installed and verified for proper operation in the test environment. Applications that were not needed for current tests were not installed to minimize potential noise and detection inaccuracies.

Some research projects (OWASP, MOTH) provided system images for fast test-case deployment. However, this was limited to only certain Web applications/sites.

The two categories of Web application test-cases used in our evaluation required different settings and configurations:

- In-house Web applications/sites: For these test-cases, we set up the required execution configurations, installed and deployed the test-cases in the test environment.
  - OWASP Broken Web Apps and MOTH were deployed using the virtual machine images alluded to previously.
  - Vulnerable versions of applications/sites for which vulnerabilities have been confirmed with reference to the CVE or OWASP database are deployed separately.
  - Note: It would be likely that the updated and/or patched versions of these vulnerable applications are already remediated from their original specified vulnerabilities. Hence, the applications' update function was disabled to prevent automatic and manual updates/upgrades.
- Existing online vulnerability-ready Web applications/sites include those at:
  - zero.webappsecurity.com,
  - demo.testfire.net,
  - testphp.acunetix.com,
  - testasp.acunetix.com,
  - testaspnet.acunetix.com,
  - crackme.cenzic.com.

4.3 Scanners

Our experiments evaluate the following scanners:

- Commercial scanners (all run on Windows-based platform)
  - IBM AppScan
  - Acunetix
  - WebSecurity
  - Netsparker Community Edition
- Open-source scanners (Windows-based and Linux-based installation)
  - Sqlmap
  - Sqlninja
  - Sqlbrute

In our experiments, we also installed HPWebInspect. However, because its evaluation version allows scanning on only one pre-defined test Website provided by the vendor (zero.webappsecurity.com), we could not use HPWebInspect on the other Web application test-casess installed in our test environment. Hence, we only used it to verify the scan results at zero.webappsecurity.com and exclude HP WebInpect in other Web application tests.

For a separate, informal evaluation of test results reliability, we performed scanning tests with sqlmap scanner which:

- Allows import of search database from a search engine (googledork).
- Scans specific web pages (links) with parameterized tests.

The comparison and analysis of test results are presented in the next section.

5 Test Results Evaluation

We observed that the SQLI-based Web application scanning results varied with different scanners. We grouped detected SQLI vulnerabilities into the following SQLI types:

- Inline SQLI (Or generic SQLI if the scanner does not state specific SQLI type)
- Inferential SQLI (Boolean-Based Blind SQL)
- AND/OR time-based blind (such as detected by sqlmap and sqlninja on testasp.vulnweb.com)
- Stack queries (detected by sqlmap on testasp.vulnweb.com)
- Related vulnerability of SQLI includes all scanner warning messages stating that SQLI may be detected and verification is needed, or Authentication Bypass Using SQL Injection (in AppScan)

The detected SQLI vulnerabilities are confirmed by scanners using different injection strings. It appears that each scanner uses its own injection string set or has generated the injection strings using different patterns (Table 1 illustrates with some results).

The scanners execute SQLI probes with different injection string sets in different trials.

Table 1. SQLI detection at testasp.vulnweb.com

SQLI type	Scanner	Injectable Location and injection string
Inline SQL Injection Error-based	Acunetix	/Login.asp?RetURL=/Default.asp? parameter: tfUPass, tfUName

		tfUPass='
		tfUPass=acunetix'"
		tfUPass=\'
		tfUName=\'
		tfUname='
		tfUname=acunetix'"

	Netsparker Community Edition	tfUName='+ (select convert(int,CHAR(95)+ CHAR(33)+CHAR(64)+CHAR(50)+CHAR(100)+ CHAR(105)+CHAR(108)+CHAR(101)+CHAR(109)+ CHAR(109)+CHAR(97)) FROM syscolumns) +'
		
		tfUPass='+ (select convert(int,CHAR(95)+CHAR(33)+ CHAR(64)+CHAR(50)+CHAR(100)+CHAR(105)+ CHAR(108)+CHAR(101)+CHAR(109)+CHAR(109)+ CHAR(97)) FROM syscolumns) +'

	IBM AppScan	tfUName=ht&tfUPass=ht%27%3B
		tfUName=ht&tfUPass=ht%uFF07
		tfUName=ht&tfUPass=ht'

	WebSecuri ty	POST http://testasp.vulnweb.com/Login.asp? RetURL=%2FDefault%2Easp%3F HTTP/1.1
		
		tfUName=Abc123&tfUPass='
		tfUName='&tfUPass=Fr3d1ee!

Inferential (a.k.a. Boolean-Based Blind SQL)	Netsparker Community Edition	tfUName=' OR 'ns'='ns
		tfUPass=' OR 'ns'='ns

	Sqlmap	tfUname=-6532' OR NOT 7373=7373--&tfUpass=123&Login=Submit
		tfUname=abc&tfUpass=123'))) AND 3880=3880-- AND ((('HIdE' LIKE 'HIdE&Login=Submit

	IBM AppScan	tfUName=ht%27+and+ %27foobar%27%3D%27foobar&tfUPass=ht
		tfUName=%27+%2B+%27%27+%2B+%27ht&tfUPass=ht
		tfUName=ht%27+and+%27foobar%27 %3D%27foobar%27+--&tfUPass=ht

AND/OR time-based blind	Sqlninja	tfUName=admin';__SQL2INJECT__&tfUPass=admin

5.1 Site discovery and exploration

In order to perform automatic scanning effectively, site structure (or site map) exploration and discovery are required. Site explorer can be a web proxy or a web crawler (spider), and they are either integrated modules of a scanner or 3rd party tool whose scanning results can be imported or called by the scanner under evaluation.

The performance of automatic SQLI scanning, and vulnerability scanning in general, depends on how detail a site explorer can obtain from a website. In our experiment with commercial tools, we observed that at least one scanner, Acunetix, did not discover two pages (/forgot1.asp, /join1.asp) in a website (zero.webappsecurity.com) while those pages were detected by other scanners and some even successfully detected SQLI vulnerability. Some other applications (e.g. Damn Vulnerability Web Application) that require user authentication into the main login page blocks the automatic crawler. For such situations, we use man-in-themiddle proxy or crawler which is proxy-capable to provide authentication support to assist in the site mapping.

Without crawling capability, proxies will have to build a site map from intercepted interactions between a client (usually the tester's host) and server. Mapping accuracy will then be dependent on the comprehensiveness and effectiveness of the monitoring. The proxy may miss some Web pages and in turn, the vulnerability scanner which uses the proxy's site map result will ignore those pages.

Recent Web applications, particularly Web service (e.g. Java-based) applications, use techniques that hide the actual pages from a crawler. End users usually see only the main Web page and the embedded interactive objects are responsible for UI tasks. Furthermore, typical crawlers/proxies can only detect a few less detailed Web pages. In such cases, a crawler and/or proxy with Web service detection capability is required. However, only a few of such tools are available. For example, we have HP WebInspect, IBM AppScan with Web service scanning. WebScarab proxy recently added Web service capture.

For blind test on Web applications, automatic site discovery should include the following:

- Traditional site exploration.
- Interactive objects detection.
- Interaction case generation or
- Interaction detection via proxy or traffic monitoring.

5.2 Variants of Scan

As alluded to previously, different scanners generate different requests/responses to the same suspected location. As a consequence,

- Request/response sets (SQLI probe sets) may be fixed and preset (e.g. the fixed sets employed by most of the commercial scanners – see Table 2).
- A probe set can be extended if a confirmed vulnerability in a specific application is added to scanner's scanning database (e.g. in Acunetix).
- Not all scanners successfully probe the same SQLI vulnerabilities at the same location.
- Not all scanners successfully probe the same SQLI types at the same location.
- Tester may use his/her own testing with favourite cheat sheets (such as [13-15] or self created) to verify the SQLI existence and confirm possible attack(s).
- In our experiments, we observed that no two scanners use same injection strings in their SQLI tests. However, the injection strings are generated for certain principal SQLI types (such as blind injection, error-based injection, or related SQLI inference) based on the valid SQL grammar.
- If we combine the scanners' request sets, we can also infer / determine different variants of SQL Injection at one location.

We suggest that SQLI research should consider the capability of generating context-aware, grammar-based injection strings based on the analysis of existing injection sets (scanners' database, SQLI cheat sheets), and the Web application's mapped structure.

Table 2. IBM AppScan regular SQLI and blind SQLI vulnerabilities injection strings

WF'SQL "Probe;A--B	and 0=7659	' or 'foobar'='foobar' --
' having 1=1--	/**/or/**/7659=7659	' or "foobar'='foobar') --
1 having 1=1--	/**/and/**/7659=7659	' and 'foobar'='foobar
\' having 1=1--	/**/and/**/0=7659	' and 'foobar'='foobar') --
) having 1=1--	'	' exec master.. xp_cmdshell 'vol'--
%a5' having 1=1—	';	'; select * from dbo.sysdatabases--
|vol	)	'; select @@version,1,1,1--
'| 'vol	\'	'; select * from master..sysmessages--
"| "vol	;	'; select * from sys.dba_users--
||vol	\"	
'+ "+ '	"'	
'+ 'somechars	"	
somechars'+'	' and 'barfoo'='foobar' --	
somechars'||'	' or 'foorbar'='foorbar	
'||'somechars	' and 'foorbar'='foobar' --	
'||'	' and 'barfoo'='foobar') --	
or 7659=7659	' and 'barfoo'='foobar	
and 7659=7659		

6 Conclusion

Our experimental evaluation and analysis on black-box Web application vulnerability scanners, focusing on SQLI vulnerability detection discovered that most contemporary scanners focus on fixed and known vulnerabilities. Hence, scanning capability is bounded within the existing known patterns of vulnerability knowledge and disclosed attacks. Besides, the indicated limitation in Web application structure discovery in pre-scanning step also restrains effective scanning. We specified two new issues that need further research in next stage. This include improving the site discovery and context-aware injection set generation for SQLI scanning and detection.

References:

[1] L. Suto. (2007). Analyzing the Effectiveness and Coverage of Web Application Security Scanners. [Online]. *2007(5-Oct)*, doi: Available: http://ha.ckers.org/blog/20071014/web-application-scanning-depth-statistics/

[2] L. Suto. (2010, 27 April 2011). Analyzing The Accuracy and Time Costs of Web Application Security Scanners. doi: Available: http://www.ntobjectives.com/files/Accuracy_and_Time_Costs_of_Web_App_Scanners.pdf

[3] "SQL Injection: Are Your Web Application Vulnerable?," SPI Labs, White Paper 2002.

[4] Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications," in the *Conference record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* Charleston, South Carolina, USA, 2006, pp. 372-382, doi: 1111037.1111070.

[5] N. Jovanovic, C. Kruegel, and E. Kirda, "Static Analysis for Detecting Taint-style Vulnerabilities in Web Applications," *Journal of Computer Security*, vol. 18, 2010, pp. 861-907, doi: 10.3233/JCS-2009-0385.

[6] "SQL Injection Signature Evasion Whitepaper: The Wrong Solution to the Right Problem," Imperva, White Paper 2005.

[7] D. Das, U. Sharma, and D. K. Bhattacharyya, "An Approach to Detection of SQL Injection Attack Based on Dynamic Query Matching," *International Journal of Computer Applications*, vol. 1, 2010, doi:

[8] C. Cerrudo, "Manipulating Microsoft SQL Server Using SQL Injection," Application Security Inc., Presentation.

[9] K. Kemalis and T. Tzouramanis, "SQL-IDS: A Specification-Based Approach for SQL-Injection Detection," in the *ACM symposium on Applied computing (SAC)*, 2008, doi: 10.1145/1363686.1364201.

[10] K.V.N.Sunitha and M. Sridevi, "Automated Detection System for SQL Injection Attack," *International Journal of Computer Science and Security (IJCSS)*, vol. 4, 2010, pp. 426-435, doi:

[11] P. Bisht, P.Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks," *TISSEC*, 2009, doi:

[12] B. Indrani and E. Ramaraj, "X-Log Authentication Technique to Prevent SQL Injection Attacks," *International Journal of Information Technology and Knowledge Management*, vol. 4, January-June 2011, pp. 323-328, doi:

[13] pentestmonkey. *MySQL SQL Injection Cheat Sheet*. Available: http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet. Last accessed: August 2011.

[14] F. Mavituna. *SQL Injection Cheat Sheet (Ver. 1.4 ed.)*. Available: http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/. Last accessed: August 2011.

[15] RSnake. *SQL Injection Cheat Sheet: ESP: for Filter Evasion*. Available: http://ha.ckers.org/sqlinjection/. Last accessed: August 2011.