IJCSET |November 2012 | Vol 2, Issue 11, 1476-1483 www.ijcset.net | ISSN:2231-0711

Protocol Based Approach on Vulnerability Detection Tools of SQLIA along with Monitoring Tools

D. Naga Swetha[1], B. Shyam Kumar[2]

Department of Information Technology
Teegala Krishna Reddy Engineering College
Hyderabad, India.
[1] swetha.tkrecit@gmail.com
[2] shyamtkrec@gmail.com

***Abstract*—SQL injection attacks pose a serious threat to the security of Web applications and web services because they can give attackers unrestricted access to databases that contain sensitive information. In this paper, we proposed a new, protocol based approach to easily identify the vulnerability taking place in web applications and web services. Our approach has both conceptual and practical advantages. We have presented an experimental evaluation of security vulnerabilities occurring in web applications and services with the protocols used for them. The different approaches to test web applications for vulnerabilities given the experimental results and statistical analysis based on today's trend. Solutions are provided for parameter tampering in SOAP protocol with detection process. As attacks are taking place on protocols, finally to take control over SQLIAs, evaluation of attacks on protocols presented with the help of monitoring tools-a new innovation.**

***Keywords*—Web services, Web applications, Protocols, SQLIA, Vulnerabilities, Vulnerability Scanners, Monitoring tools.**

I. INTRODUCTION

More and more of the applications we use every day are moving online nothing but internet. The internet carries an extensive range of information resources and services, such as the inter-linked hypertext documents of the World Wide Web (www) and the infrastructure to support e-mail. Basically, internet is an insecure channel for exchanging information. A new challenge is posed on software engineers for testing web applications and web services against web security threats. Scattered data over the web or internet has given a chance to the hackers to gain access on the data from various data bases. For example, if a hacker hacks any individual user's credentials, he can misuse the user's account information for various purposes and causes huge amount of harm to the concerned user. Most of the database applications such as banking, online shopping etc. often consist of crucial and valuable information. Mostly, many of the database applications dynamically generate commands in the database language that is SQL-Structured query language. It is a computer language aimed to store, manipulate, and query data stored in relational databases. We use SQLcommands/Queries for storing, retrieving and manipulating data in a relational database. One particular type of attack, which gives the attackers a way to gain complete access to the databases underlying web applications, is an SQL injection attack (SQLIA). By using this attack, hackers can easily alter or may delete the information stored in the databases. Insufficient input validation is the major reason for SQL injection vulnerabilities.

In this paper, we present all the different types of SQL injection attacks along with the focusing areas of attackers whose intension is to hack the crucial data of the concerned user. Along with this information, a new proposal of comparison tools of all different categories has presented with protocol suite. All the tools presented in this paper play their individual role with respect to their mentioned areas. To draw the attention of attackers on web applications and web services, we need to find the vulnerabilities in web applications that are done with SQL injections. To overcome this problem and to provide information to the users we presented all the attacks from the attackers and detection of attacks also in this paper.

II. MOTIVATION: SQL INJECTION ATTACKS

A SQL Injection attack can occur when a web application utilizes user-supplied data without proper validation or encoding as part of a command or query. The specially crafted user data tricks the application into executing unintended commands or changing data[1]. Attackers provide specially crafted input data to the SQL interpreter and trick the interpreter to execute unintended commands. With over 20% of all web vulnerabilities being attributed to SQL Injection, this is the 2nd most common software vulnerability and having the ability to find and prevent SQL injection should be top of mind for web developers and security personnel.

2.1 Web application and SQL injection attack (SQLIA) with example:

A. Web Application Architecture:

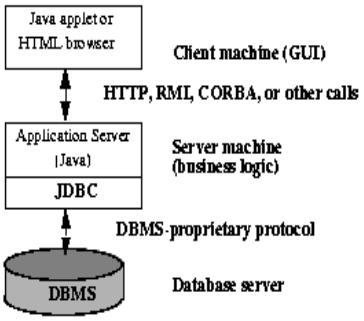


Fig. 1 3-tier web application architecture

In the architectures shown in Fig 1, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user.

B. SQL Injection Attack (SQLIA) with example:

An SQL Injection Attack (SQLIA)[2] occurs when an attacker changes the developer's intended structure of an SQL command by inserting new SQL keywords or operators.

Let's have a look at the SQL example given below.

String SQLQuery="SELECT Username, Password FROM users WHERE Username=' " + Username + " ' AND Password=' " + Password + " ' " ;

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(SQLQuery);

while (rs.next()) { … }

If an attacker provides 'or 0=0' as the username and password, then the query will be constructed as: *String SQLQuery ="SELECT Username, Password FROM users WHERE Username=" or 0=0" AND Password=" or 0=0";*

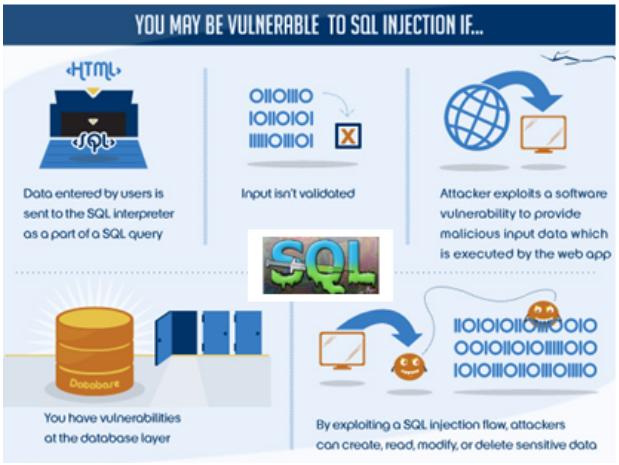


Fig. 2 SQLIA Example

SQL injection can be prevented by adopting an input validation technique in which user input is authenticated against a set of defined rules for length, type, and syntax and also against business rules.

2.2 Classification Parameters and Mechanisms in SQLIA:

The attacking vector contains five main sub-classes depending on the technical aspects of the attack's deployment.

The common mechanisms we explain in this section are as follows:

Injection through user input: In this case, attackers inject SQL commands by providing suitably crafted user input.

Injection through cookies: Cookies are files that contain state information generated by Web applications and stored on the client machine. When a client returns to a Web application,

cookies can be used to restore the client's state information.

TABLE I
CLASSIFICATION PARAMETERS

Classification parameters	Methods	Techniques/ Implementation	
Intent	Identifying injectable parameters	see 'Input type of attacks'	
	Extracting Data		
	Adding or Modifying Data		
	Performing Denial of Service		
	Evading detection		
	Bypassing Authentication		
	Executing remote commands		
	Performing privilege escalation		
Input Source	Injection through user input	Malicious strings in Web forms	URL: GET- Method
			Input filed(s): POST- Method
	Injection through cookies	Modified cookie fields containing SQLIA	
	Injection through server variables	Headers are manipulated to contain SQLIA	
	Second-order injection	Frequency-based Primary Application	
		Frequency-based Secondary Application	
		Secondary Support Application	
		Cascaded Submission Application	
Input type of attacks, technical aspect	Classic SQLIA	Piggy-Backed Queries	
		Tautologies	
		Alternate Encodings	
		Illegal/ Logically Incorrect Queries	
		UNION SQLIA	
		Stored Procedures SQLIA	
	Inference	Classic Blind SQLIA	Conditional Responses
			Conditional Errors
			Out-Of-Band Channeling
		Timing SQLIA	Double Blind SQLIA(Time-delays/ Benchmark attacks)
			Deep Blind SQLIA (Multiple statements SQLIA)
	DBMS specific SQLIA	DB Fingerprinting	
		DB Mapping	
	Compounded SQLIA	Fast-Fluxing SQLIA	

Injection through server variables: Web applications use these server variables in a variety of ways, such as logging usage statistics and identifying browsing trends.

Second-order injection: In second-order injections, attackers seed malicious inputs into a system or database to indirectly trigger an SQLIA when that input is used at a later time.

III. OUR APPROACH

3.1 Different types of SQL Injection Attacks:

This section gives a brief information regarding the various types of SQL injection attacks took place in web applications.

A. Tautologies:

Tautology-based attack [4] is to inject code in one or more conditional statements so that they always evaluate to true.

Example:

In this example attack, an attacker submits " ' or 1=1 - -".

The Query for Login mode is:

SELECT * FROM user info WHERE loginID='' or 1=1 - - AND pass1=''

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology the query evaluates to true for each row in the table and returns all of them. The application would invoke method user_main.aspx and to access the application.

B. Union Query:

The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

Example: An attacker could inject the text

"' UNION SELECT pass1 from user_info where LoginID='secret - -" into the login field, which produces the following query:

SELECT pass1 FROM user_info WHERE loginID='' UNION SELECT pass1 from user_info where LoginID='secret' -- AND pass1=''
The database takes the results of these two queries, unions them, and returns them to the application. In many applications, the effect of this operation is that the value for "pass1" is displayed along with the account information.

C. Piggy-Backed Queries:
In this attack type, an attacker tries to inject additional queries into the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first.
Example: If the attacker inputs "drop table users - -" into the *pass* field, the application generates the query:
SELECT accounts FROM users WHERE login='doe' AND pass= ' ' ,drop table users -- ' AND pin=123

The solution like simply scanning for a query separator is not an effective way to prevent this type of attack.

IV. OUR IMPLEMENTATION: PROTOCOL PERFORMANCE

4.1 Vulnerability Detection Techniques for Web Services

4.1.1 Web Vulnerability Scanners:
Web services provide a simple interface between a provider and a consumer and are supported by a complex software infrastructure, which typically includes an application server, the operating system and a set of external systems(eg. databases).[9] Security vulnerabilities like SQL injection and XPath injection attacks take advantage of improper coded applications to change SQL commands that are sent to the database.
4.1.2 Approaches to test Web Applications for Vulnerabilities:
White Box testing: It examines the internal source code of the web application. Because of the complexity of code, we may not find all security flaws in the code.
Black Box testing: It examines the execution of the application in search for vulnerabilities.
4.1.3 Vulnerability Types in Web Services:
XPath Injection: It is possible to modify an XPath query to *"be parsed in a way differing from the programmer's intention"*. Attackers may gain access to information in XML
documents[9].
Code Execution: It is possible to manipulate the application inputs to trigger server-side code execution[9]. An attacker can exploit this vulnerability to execute malicious code in the server machine.
Buffer Overflow: It is possible to manipulate inputs in such a way that causes buffer allocation problems, including overwriting of parts of the memory[9]. An attacker can exploit this causing Denial of Service or, in worst cases, *"alter application flow and force unintended actions"*.
Username/Password Disclosure: The web service response contains information related to usernames and/or passwords[9]. An attacker can use this information to get access to private data.
Server Path Disclosure: The response contains a fully qualified path name to the root of the server storage system. [9]. An attacker can use this info to discover the server file system structure and devise other security attacks.

TABLE II
OVERALL RESULTS

Vulnerability Types	VS1.1		VS1.2		VS2		VS3	
	# Vuln.	# WS	# Vuln.	# WS	# Vuln.	# WS	# Vuln.	# WS
SQL Injection	217	38	225	38	25	5	35	11
XPath Injection	10	1	10	1				
Code Execution	1	1	1	1				
Possible Parameter Based Buffer Overflow							4	3
Possible Username or Password Disclosure							47	3
Possible Server Path Disclosure							17	5
Total	**228**	**40**	**236**	**40**	**25**	**5**	**103**	**22**

Scanners classify these situations as low importance security issues.

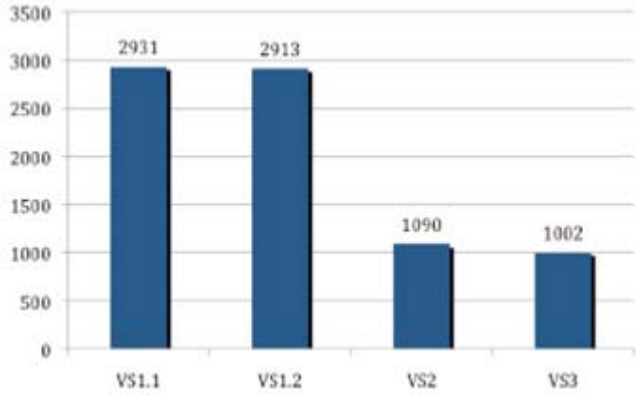


Fig. 3 Application Errors detected

In the above graph, Scanners VS1.1 and VS1.2 detected a code execution vulnerability. This is a particularly critical vulnerability that allows attackers to execute code in the server. VS3 was the only one pointing vulnerabilities related to buffer overflow, username and password disclosure, and server path disclosure.

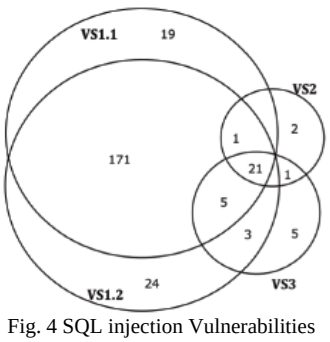


Fig. 4 SQL injection Vulnerabilities

The above figure shows the intersection areas of the circles which represent the number of vulnerabilities detected by more than one scanner. It clearly shows that the four scanners detected different sets of SQL Injection vulnerabilities and the differences are considerable, pointing again to relatively low coverage of each vulnerability scanner individually.

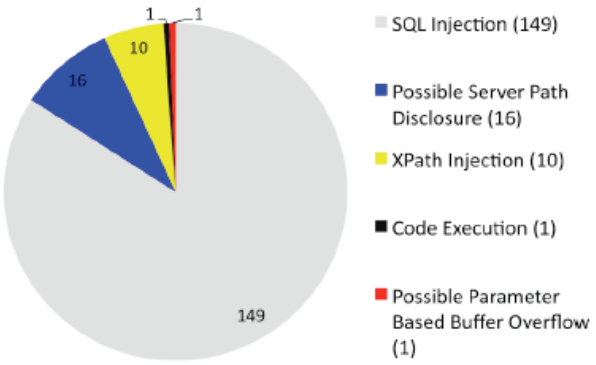


Fig. 5 Vulnerabilities distributed per type

The above figure shows the final distribution of vulnerabilities
per type, after removing the confirmed false positives but including the doubtful cases (i.e., optimistic evaluation of the scanners)

4.2 SOAP: Protocol used for Web Vulnerability Scanner

Simplified Object Access Protocol (SOAP) is a specification that enables applications to communicate with other applications. It provides a framework for connecting Web sites and applications to create *Web services.*

4.2.1 SOAP Architecture:

The above figure shows the overall architecture of a generic system built using SOAP. This system uses HTTP protocol to pass the SOAP message between the client and the server. The client application calls a client-side proxy object using its native RPC protocol. The proxy object uses an XML parser to convert the call into a SOAP packet. This SOAP packet is then transmitted over the Internet/Intranet to the web server using the HTTP protocol. The Web server handles the URL connection point of the remote service, and launches a SOAP translator which may be an ASP page, an ISAPI extension, a CGI program, a Perl script, etc. This translator uses a local XML parser to parse out the object name, method name and parameter values from the SOAP package. It uses these values to call the particular method of the server object by the local ORPC protocol, and packages the results into a response SOAP packet. This response is unpackaged by the proxy and presented to the client.

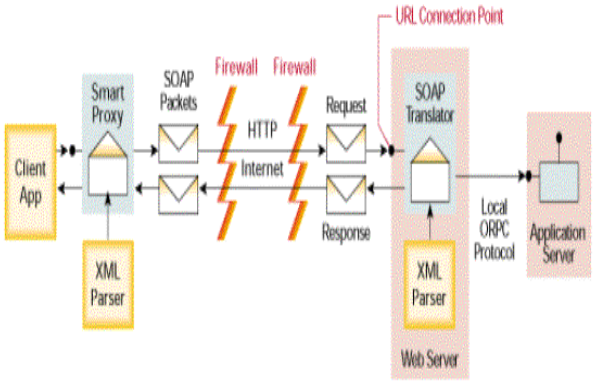


Fig. 6 SOAP Architecture

4.2.2 SQL Injection through SOAP Parameter Tampering:

An attacker modifies the parameters of the SOAP message that is sent from the service consumer to the service provider to initiate a SQL injection attack. On the service provider side, the SOAP message is parsed and parameters are not properly validated before being used to access a database in a way that does not use parameter binding, thus enabling the attacker to control the structure of the executed SQL query. This pattern describes a SQL injection attack with the delivery mechanism being a SOAP message.

Attack Execution Flow:

A. *Detect Incorrect SOAP Parameter Handling:*

TABLE III
DETECTING SOAP PARAMETER TAMPERING

ID	Attack Step Technique Description	Environments
1	The attacker tampers with the SOAP message parameters by injecting some special characters such as single quotes, double quotes, semi columns, etc. The attacker observes system behavior.	env-Web

Indicators

ID	Type	Indicator Description	Environments
1	Positive	SOAP messages are used as a communication mechanism in the system	env-Web

Outcomes

ID	Type	Outcome Description
1	Success	Any indication that the injected input is causing system trouble (e.g. stack traces are produced, the system does not respond, etc.) then the attacker may come to conclude that the system is vulnerable to SQL injection through SOAP parameter tampering.

The attacker tampers with the SOAP message parameters and looks for indications that the tamper caused a change in behavior of the targeted application.

B. Inject SQL via SOAP Parameters:

The attacker injects SQL via SOAP parameters identified as vulnerable during Explore phase to launch a first or second order SQL injection attack.

TABLE IV
DETECTING VULNERABILITY

Attack Step Techniques

ID	Attack Step Technique Description	Environments
1	An attacker performs a SQL injection attack via the usual methods leveraging SOAP parameters as the injection vector. An attacker has to be careful not to break the XML parser at the service provider which may prevent the payload getting through to the SQL query. The attacker may also look at the WSDL for the web service (if available) to better understand what is expected by the service provider.	env-Web

Outcomes

ID	Type	Outcome Description
1	Success	Attacker achieves goal of unauthorized system access, denial of service, etc.
2	Failure	Attacker unable to exploit SQL Injection vulnerability.

Security Controls

ID	Type	Security Control Description
1	Detective	Search for and alert on unexpected SQL keywords in application logs (e.g. SELECT, DROP, etc.).
2	Preventative	Input validation of SOAP parameter data before including it in a SQL query
3	Preventative	Use parameterized queries (e.g. PreparedStatement in Java, and Command.Parameters.Add() to set query parameters in .NET)

C. Solutions and Mitigations:

Properly validate and sanitize user input at the service provider. Ensure that prepared statements or other mechanism that enables parameter binding is used when accessing the database in a way that would prevent the attacker's supplied data from controlling the structure of the executed query. At the database level, ensure that the database user used by the application in a particular context has the minimum needed privileges to the database that are needed to perform the operation. When possible, run queries against regenerated views rather than the tables directly.

4.3 Vulnerability Detection Techniques for Web Application

4.3.1 AJECT: Vulnerability Detection Tool

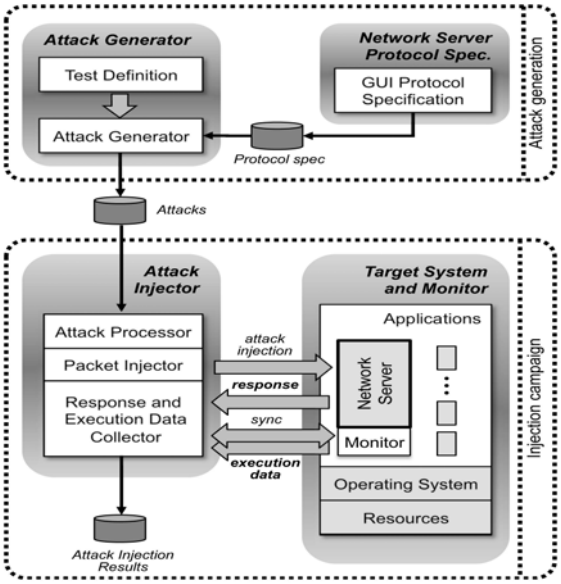


Fig. 7 AJECT Architecture

The Attack inJECtion Tool (AJECT) was designed to look for vulnerabilities in network server applications, although it can also be utilized with local daemons. AJECT does not need the source code of the server to perform the attacks, i.e., it treats the server as a *black box*. AJECT has to obtain a specification of the protocol utilized in the communication with the server.

In the above shown architecture of AJECT, the *Target System* is the entire software and hardware components that comprise the target application and its execution environment. The *Network Server* is typically a service that can be queried remotely from client programs (e.g., a mail or FTP server). The Network Server Protocol Specification is a graphical user interface component that supports the specification of the communication protocol used by the server. This specification is utilized by the Attack Generator to produce a large number of test cases. The Attack Injector is responsible for the actual execution of the attacks by transmitting malicious packets to the server.

4.3.2 Example: A Software component with vulnerable points

Architecture view:

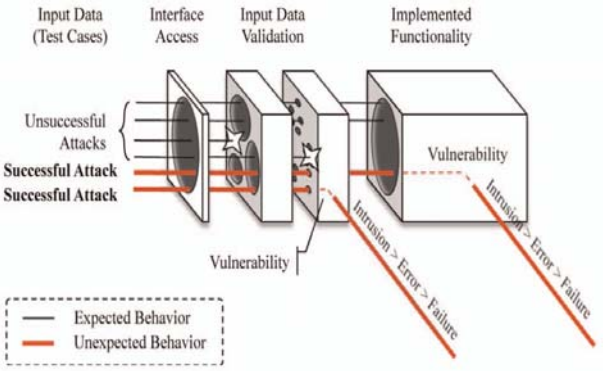


Fig. 8 Architecture view of a software component with vulnerabilities

The external access to the component is provided through a known Interface Access, which receives the input arriving in network packets or disk files, and eventually returns some output. Whether the component is a simple function that performs a specific task or a complex system, its intended functionality is, or should be, protected by Input Data Validation layers. By accessing the interface, an adversary may persistently look for vulnerabilities by stressing the component with unusual forms of interaction, such as sending wrong message types or opening malformed files.

4.4 Protocols used for AJECT tool

A. POP(Post Office Protocol): The Post Office Protocol is designed to allow a workstation to dynamically access a mail drop on a server host. POP3 allows a workstation to retrieve mail that the server is holding for it. POP3 is not designed to provide extensive manipulation operations of mail on the server; which are done by a more advanced protocol IMAP4. POP3 uses TCP as the transport protocol.

B. SMTP-Simple Mail Transfer Protocol: Simple Mail Transfer Protocol (SMTP) is a protocol designed to transfer electronic mail reliably and efficiently. An important feature of SMTP is its capability to transport mail across networks, usually referred to as “SMTP mail relaying”.

C. IMAP & IMAP4(version 4): Internet Message Access Protocol (IMAP) is a method of accessing electronic mail or bulletin board messages that are kept on a mail server. IMAP includes operations for creating, deleting and renaming mailboxes, checking for new messages and many more.

D. MIME (S-MIME): Multipurpose Internet Mail Extensions and Secure MIME. MIME is a very flexible format, permitting one to include virtually any type of file or document in an email message. A secure version of MIME, S/MIME (Secure/Multipurpose Internet Mail Extensions), is defined to support encryption of email messages.

4.5 Attacks on Protocols used for AJECT Tool

4.5.1 Attack on IMAP:

In this case, command injection is done over the IMAP server so they must follow the format and specifications of this protocol. The webmail applications communicate with the IMAP server to carry out their operations and that's the reason why they are more vulnerable to this kind of attack. During user authentication the webmail application transmits credentials to the IMAP server, so it's possible the IMAP Injection takes place without the need of having a valid account in the application, exploiting the authentication mechanism of the IMAP server.

Example: Let's see an example of IMAP Injection by exploiting the functionalities of reading a message.

Assume that the webmail application uses the parameter "message_id" to store the identifier of the message that the user wants to read. When a request containing the message identifiers is sent the request would appear as:

http:// <webmail>/read_email.php?message_id=<number>

Suppose that the webpage "read_email.php", responsible for showing the associated message, transmits the request to the IMAP server without performing any validation over the value <number> given by the user. The command sent to the mail server would look like this:

FETCH <number> BODY[HEADER]

In this context, a malicious user could try to conduct IMAP Injection attacks through the parameter "message_id" used by the application to communicate with the mail server.
For example, the IMAP command "CAPABILITY" could be injected using the next sequence:
http://<webmail>/read_email.php?message_id=1 BODY[HEADER]%0d%0aZ900CAPABILITY%0d%0a Z901 FETCH 1
This would produce the next sequence on IMAP commands in the server:
FETCH 1 BODY[HEADER] Z900 CAPABILITY
Z901 FETCH 1 BODY[HEADER]
So the page returned by the server would show the result of the command "CAPABILITY" in the IMAP server:
* CAPABILITY IMAP4rev1 CHILDREN NAMESPACE THREAD=ORDEREDSUBJECT THREAD=REFERENCES SORT QUOTA ACL ACL2=UNION Z900 OK CAPABILITY completed
4.5.2 Attack on SMTP:
In this case, the command injection is performed to the SMTP server. Due to the operations permitted by the application using the SMTP protocol we are basically imitated to sending e-mail. The use of SMTP Injection requires that the user be authenticated previously, so it is necessary that the user have a valid webmail account.
Example: Let's see an example of the SMTP Injection technique through the parameter that holds the subject of a message.
A typical request for e-mail sending would look like this:
POST http://<webmail>/compose.php HTTP/1.1----------134475172700422922879687252
Content-Disposition: form-data; name="subject" SMTP Injection Example----------134475172700422922879687252
Which would generate the next sequence of SMTP commands:
MAIL FROM: <mailfrom> RCPT TO: <rcptto> DATA Subject: SMTP Injection Example ...
If the application doesn't correctly validate the value in the parameter "subject", an attacker could inject additional SMTP commands into it:
POST http://<webmail>/compose.php HTTP/1.1--------134475172700422922879687252
Content-Disposition: form-data; name="subject"
SMTP Injection Example
MAIL FROM: notexist@external.com
RCPT TO: user@domain.com
DATA Email data-------134475172700422922879687252 ...
The commands injected above would produce a SMTP command sequence that would be sent to the mail server, which would include the MAIL FROM, RCPT TO and DATA commands as shown here:
MAIL FROM: <mailfrom>
RCPT TO: <rcptto>
DATA
Subject: SMTP Injection Example
MAIL FROM: notexist@external.com
RCPT TO: user@domain.com
DATA Email data

V. EMPIRICAL EVALUATION

5.1 Evaluation of Attacks on Protocols by using Monitoring Tools:
The monitoring tool can provide more accurate identification of unexpected behavior. Its implementation requires 1) access to the low-level process and resource management functions of the target system and 2) synchronization between the injector and the monitor for each test case execution.
In order to avoid potential monitoring restrictions and to support as many target systems as possible, three alternative monitoring components were developed. They are presented in the following way.
A. Deep Monitor: The Deep monitor observes the target application's flow of control, while keeping a record of the amount of allocated system resources. The tracing capabilities are achieved with the PTRACE family functions to intercept any system calls and signals received by the server process Since the deep monitor is OS-dependent, it can only be used in UNIX-based systems.

B. Shallow Monitor: The Shallow monitor is platform independent. It control the server and collect the return status code after the completion of every test case. At this moment, both implementations of deep and shallow monitors have a limitation that they cannot monitor background processes as they immediately detach themselves from the main process.

C. Remote Monitor: The Remote monitor infers the server's behavior through passive and external observation. It resides in the injector machine and collects information about the network connection between the *injector and the server*. After every test case execution, the monitor closes and reopens the connection with the server, signaling the server's execution as failed if some communication error arises.
5.2 Green SQL - A New Innovation Of Monitoring Tools
It is a database firewall engine used to protect open source databases from SQL injection attacks. It works in proxy mode. Application logics is based on evaluating of SQL commands using risk score factors, as well as blocking of sensitive commands.
A. GreenSQL Database Activity Monitoring (DAM) is a powerful solution that independently monitors and audits all database activity across multiple database platforms. Using administrative access, we can easily define audit trail policy for all of your sensitive tables and/or columns, and see a "before and after" view of all changes made.
Architecture of GreenSQL:

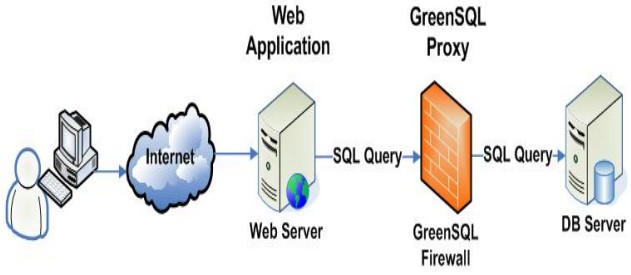


Fig. 9 Green SQL architecture

In a client/server model, GreenSQL is placed between the database(s) and the clients. It acts as a reverse proxy and analyzes the SQL commands passed to the server. Based on rules, GreenSQL forward or drop the request. The logic is based on evaluation of SQL commands using a risk scoring matrix as well as blocking known db administrative commands (DROP, CREATE, etc).

TABLE-V
GREEN SQL DASHBOARD

TABLE-V1
DISPLAYING WHITELIST OF APPROVED QUERIES

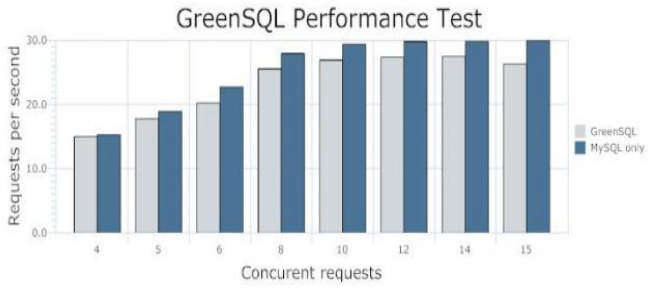
Whitelist of approved queries

ID	Time	Listener	DB	Pattern
206	2008-09-12 16:23:48	Default Proxy	joomla	select count(*) from jos_hwdvidsgroup_membership as a left join jos_hwdvidsgroup...
205	2008-09-12 16:23:48	Default Proxy	joomla	select username from jos_users where id = ?
204	2008-09-12 16:23:48	Default Proxy	joomla	select count(*) from jos_hwdvidsfavorites as a where a.userid = ? and a.videoid = ?
203	2008-09-12 16:23:48	Default Proxy	joomla	select a.* from jos_hwdvidsvideos as a where a.id = ?
202	2008-09-12 16:23:48	Default Proxy	joomla	update jos_hwdvidsvideos set number_of_views = ? where id = ?

If the query is considered illegal - whitelist is check. If it was found in the whitelist, it will be redirected to genuine MySQL server. If it was not found, an empty result set will be send to application.

B. Performance Test:

We tested web application, that makes heavy usage of database **(**82 SQL requests per page request), we get to performance decrease of 2-12 % . For high load website, web application speed will be decreased slightly.



Concurent Requests	4	5	6	8	10	12	14	15
GreenSQL	15.01	17.80	20.23	25.57	26.91	27.37	27.52	26.31
MySQL only	15.29	18.95	22.74	27.93	29.41	29.82	29.87	29.93

Fig. 10 Graphical representation of Green SQL performance Test

Performance test details:

During the test we perform measurement of the response time of the website's home page. When homepage is accessed, 82 SQL queries are executed. Apache Bench tool is used to measure response time of the web site and number of requests executed per second. It was executed like this:

ab -n 400 -c 10 hxxp://test-website.com/

- **-c** specifies number of concurrent requests
- **-n** specifies total number of requests to perform

In this test 400 requests are executed playing with the **-c** argument (number of concurrent requests).

VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented a classification parameters and mechanisms in SQLIA. Firstly, we focused on vulnerabilities taken place in all different kind of web applications. A survey has been done to identify all the protocols used for vulnerability detection. We also found how attacks are injected to web applications with the help of protocols and provided countermeasures to avoid and take complete control on the attacks.

Future evaluation work should focus on higher heuristic detection techniques in practice. Empirical evaluations such as evaluation of attacks on protocols by using monitoring tools presented in this paper would allow for comparing the performance of all the monitoring tools used for different vulnerability detection tools.

REFERENCES

[1] W. G. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In *Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005)*, Long Beach, CA, USA, Nov 2005.

[2] W. G. Halfond and A. Orso. Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks. In *Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005)*, pages 22–28, St. Louis, MO, USA, May 2005.

[3] C. Anley. Advanced SQL Injection In SQL Server Applications. White paper, Next Generation Security Software Ltd., 2002.

[4] G. Wassermann and Z. Su. An Analysis Framework for Security in Web Applications. In Proc. of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS 2004), pages 70–78, Oct. 2004.

[5] Y. Xie and A. Aiken. Static Detection of Security Vulnerabilities in Scripting Languages. In Proceedings of the 15th USENIX Security Symposium, July 2006.

[6] F. Valeur, D. Mutz, and G. Vigna. A Learning-Based Approach tothe Detection of SQL Attacks. In Proc. of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), Vienna, Austria, Jul. 2005.

[7] W. G. Halfond, J. Viegas, and A. Orso. A Classification of SQL-Injection Attacks and Countermeasures. In Proc. of the Intl.Symposium on Secure Software Engineering, Mar. 2006.

[8] Stuttard, D., Pinto, M., "The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws", Wiley, ISBN-10: 0470170778, October 2007.

[9] Fonseca, J., Vieira, M., Madeira, H., "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks", 13th IEEE Pacific Rim Dependable Computing Conference (PRDC 2007), Melbourne, Victoria, Australia, December 2007

[10] Using Web Security Scanners to Detect Vulnerabilities in Web Services Marco Vieira, Nuno Antunes, and Henrique Madeira *CISUC, Department of Informatics Engineering University of Coimbra – Portugal*

[11] A heuristic-based approach for detecting SQL-injection vulnerabilities in web applications By Angelo CiampaUniv. Of

Sannio, ItalyCorrado Aaron VisaggioUniv. Of Sannio, ItalyMassimiliano Di PentaUniv. Of Sannio, Italy.

[12] A Survey of SQL Injection Defense Mechanisms By Kasra Amirtahmasebi, Seyed Reza Jalalinia and Saghar Khadem, Chalmers University of Technology, Sweden.

AUTHORS

D. NAGA SWETHA[1] done B.Tech in Computer Science and Engineering. Working as ASSISTANT PROFESSOR in Department of Information Technology.

B. SHYAM KUMAR[2], HEAD OF THE DEPT of Information Technology, done M.Tech in Software Engineering having 5 years of teaching and 2 years of Industry experience.