Journal of Computer and Communications, 2014, 2, 1-9
Published Online June 2014 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2014.28001

A Survey of SQL Injection Attack Detection and Prevention

Khaled Elshazly[1], Yasser Fouad[2], Mohamed Saleh[3], Adel Sewisy[4]

[1]Demonstrator of Computer Science, Information System Institute, Suez, Egypt
[2]Lecturer of Computer Science, Faculty of Science, Suez University, Suez, Egypt
[3]Assistant Professor, Head of Department of Mathematics, Faculty of Science, Suez University, Suez, Egypt
[4]Professor of Computer Science, Faculty of Computers & Information, Assiut University, Assiut, Egypt
Email: yasserfrb@suezuniv.edu.eg

Received 21 April 2014; revised 16 May 2014; accepted 23 May 2014

Abstract

Structured Query Language Injection Attack (SQLIA) is the most exposed to attack on the Internet. From this attack, the attacker can take control of the database therefore be able to interpolate the data from the database server for the website. Hence, the big challenge became to secure such website against attack via the Internet. We have presented different types of attack methods and prevention techniques of SQLIA which were used to aid the design and implementation of our model. In the paper, work is separated into two parts. The first aims to put SQLIA into perspective by outlining some of the materials and researches that have already been completed. The section suggesting methods of mitigating SQLIA aims to clarify some misconceptions about SQLIA prevention and provides some useful tips to software developers and database administrators. The second details the creation of a filtering proxy server used to prevent a SQL injection attack and analyses the performance impact of the filtering process on web application.

Keywords

SQL Injection, Database Security, Attack, Authentication

1. Introduction

With the development of World Wide Web the organizations are beginning to get more sophisticated about how they employ their website. Nowadays, the web has become very essential need of our society. But with the wide spread uses of Internet, some malicious users begin the work in negative direction which harms the website of the organizations and these users are referred as cyber criminal or website attacker. SQL Injection Attacks

(SQLIA) are one of the top threats for web application security, and SQL injections are one of the most serious vulnerability types. SQLIA are easy to learn and exploitable, so this method of attack is easily used by attackers. Also many major and traditional security systems having different security layers like firewall, encryption, intrusion detection systems, Antivirus and anti malware are not able to detect this type of attack. SQLIA techniques have become more common, more ambitious, easy to learn/implement, and increasingly sophisticated, so there is a need to find an effective and feasible solution for this problem in the computer security community.

SQL is a textual relational database language. There are many varieties of SQL; however, the differences among the various dialects are minor. SQL functions fit into two categories:

- Data definition language (DDL): used to create tables and define access rights.
- Data manipulation language (DML): SQL commands that allow one to insert, update, delete, and retrieve data within database tables.

The typical unit of execution of SQL is the query, which is a collection of statements that typically return a single result. SQL injection is a way to attack a database through a firewall by taking advantage of non-validated SQL vulnerabilities. It is a method by which the parameters of a Web based application are modified in order to change the SQL statements that are passed to a backend database. An attacker is able to insert a series of SQL statements into a query by manipulating the data input, for example, by adding a single quote (') to the parameters. It is possible to cause a second query to be executed with the first [1].

Programmers often chain SQL commands together with user-provided parameters, and can therefore embed SQL commands inside these parameters. This is known as dynamic SQL. It must be noted that dynamic SQL must be used in the application or SQL injection is not possible. SQL injection has been described as a “code hole” that is as serious as any IIS hole [2] [3].

An attack against a database using SQL Injection could be motivated by three primary objectives:

1) To steal data from a database from which the data should not normally be available.

2) To obtain system configuration data that would allow an attack profile to be built. One example of this would obtain all of the database password hashes so that passwords can be brute-forced.

To gain access to an organization’s host computers via the machine hosting the database can be done using package procedures and 3GL language extensions that allow O/S access [4].

2. Background Examples

SQLIA occurs when an attacker causes the web application to generate SQL queries that are functionally different from what the user interface programmer intended.

For example, consider an application dealing with author details.

A typical SQL statement looks like this:

Select id, first name, last name from authors;

This statement will retrieve the “id”, “forename” and “surname” columns from the “authors” table, returning all rows in the table. The “result set” could be restricted to a specific “author” using “where” clause.

Select id, first name, last name from authors where first name = “James” and last name = “Baker”;

An important point to note here is that the string literals “James” and “Baker” are delimited with single quotes. Here the literals are given by the user and so they could be modified. They become the vulnerable area in the application. Now, to drop the table called “authors”, a vulnerable literal can be injected into the statement as given below.

First name: “Jam”; drop table authors—last name:

Now the statement becomes:

Select id, first name, last name from authors where first name = “Jam”; drop table authors—and last name = “ ”;

This is executed since the first name ends with delimiter and — is given at the end of the input, all other command following the — is neglected. The output of this command is the deletion of the table named “authors”, which is not the intended result from a server database.

3. Related Work and Observations of SQLIA

There are four main categories of SQLIA against databases:

1) *SQL Manipulation*: manipulation is process of modifying the SQL statements by using various operations such as UNION. Another way for implementing SQL Injection using SQL Manipulation method is by changing the where clause of the SQL statement to get different results.

2) *Code Injection*: Code injection is process of inserting new SQL statements or database commands into the vulnerable SQL statement. One of the code injection attacks is to append a SQL Server EXECUTE command to the vulnerable SQL statement. This type of attack is only possible when multiple SQL statements per database request are supported.

3) *Function Call Injection*: Function call injection is process of inserting various database function calls into a vulnerable SQL statement. These function calls could be making operating system calls or manipulate data in the database.

4) *Buffer Overflows*: Buffer overflow is caused by using function call injection. For most of the commercial and open source databases, patches are available. This type of attack is possible when the server is un-patched.

4. Current Situations

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save as command, and use the naming convention prescribed by your journal for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper.

According to Huang [5], many Web applications go through rapid development phases with extremely short turnaround time, making it difficult to eliminate security vulnerabilities such as SQL injection and cross-site scripting [6]. Offers the following tips for preventing SQL injection:

1) Validate all user input before transmitting it to the server;

2) Permit only minimally privileged accounts to send user input to the server;

3) Run SQL Server itself with the least necessary privileges;

The presentation by [7] at a Black Hat USA 2004 convention outlines automated blind SQL injection techniques. He mentions that string comparison is suitable for error based SQL injection but not blind SQL injection. He also mentions that there are three kinds of SQL injection:

- Redirecting and reshaping a query
- Error message based
- Blind SQL injection

[8] provides a good background into the problem of SQL injection. It puts the whole problem into context. The site provides explanations of the components of SQL injection strings and the syntax choices. The examples include SQLIA, creating a secure data access component using Java's regular expressions.

[9] provides concise examples of SQL injection and database error messages as well as methods on how to prevent SQL injection. The white paper by [4] covers research into SQL injection as it applies to Microsoft Internet Information Server/Active Server Pages/MS SQL Server platform. It addresses some of the data validation and database lockdown issues that are related to SQL injection into applications. The paper provides examples of SQLIA and gives some insight into .asp login code and query error messages used to exploit databases.

[10] goes through worked examples of SQLIA in his white paper on Detecting SQL Injection in Oracle. It focuses on detecting SQL injection by auditing the error message log files. It attempts to highlight the fact that during a hacking attempt, the error messages leave a trail that can help expose the vulnerabilities of the database being attacked.

[11] goes through some common SQL injection techniques and proposes a solution to the problem. The paper provides a list of database tables that are useful to SQL injection in MS SQL Server, MS Access and Oracle. It also provides examples of SQL injection using select, insert, union, stored procedures. The examples work with a web service that returns information to the user. This paper deals primarily with the structure of the SQL injection commands and guides to overcoming possible errors returned by the database. It should be noted that SQL injection can still occur if there is no feedback to the client. So, one could create a new valid user in a database without receiving errors and then log on.

[12] points out that the scanner is restricted to looking for classes of vulnerabilities such as SQL injection or cross site scripting. The reason for this being that the benefit of known security issues is lost because the remote scanner does not have access to the source code. There is no way to provide everyone with the minimum privi-

leges necessary. Thus the paper explores some simple techniques in extracting the logging and trace data that could be used for monitoring.

This paper is an extension of a two-part paper on investigating the possibilities for an Oracle database administrator to detect SQL injection. This paper provides many scripts on SQL injection and extracting logs.

5. Existing Products

SQL injection is not a new problem. The date of its discovery is uncertain. However, in the last few years, SQLIA have been on the rise [13]. Applications have still proven to be vulnerable despite all efforts to limit information returned to the client. There are a few applications that have been developed by companies in an effort to provide a solution to this problem.

Some have been outlined below:

- *Secure Sphere*

Uses advanced anomaly detection, event correlation, and a broad set of signature dictionaries to protect web applications and databases. It also uses error responses from the same user to identify an attack [14].

- *Mod Security*

Is an open source intrusion detection engine for web applications, which may provide helpful tips on how to detect SQL injection. [15] has developed Mod Security for Java which is a Servlet 2.3 filter that stands between a browser and the application, monitors requests and responses as they are passing by, and intervenes when appropriate in order to prevent attacks.

- *Airlock*

Combines secure reverse proxy with intrusion prevention, content filtering, user authentication enforcement, and application-level load balancing and failover (Seclutions' Airlock was awarded the Swiss Technology Award 2003) [16].

- *VIPER tool for penetration testing*

According to Angelo Ciampa, Corrado Aaron Visaggio and Massimiliano Di Penta, they have suggested a tool called Viper to perform penetration testing of Web applications. This tool relies on a knowledge base of heuristics that guides the generation of the SQL queries. This tool first identifies the hyperlink structure and its input from [17].

- *SQLr and Practical Protection mechanism*

S. W. Boyd and A. D. Keromytis has suggested the practical protection mechanism for preventing SQLIA against web server. This tool uses SQL randomized query CGI application and detect and correct the queries injected into the code [18].

- *Green SQL*

Is a free Open Source database firewall that sits between the web server and the database server and is used to protect databases from SQL injection attacks. The logic is based on evaluation of SQL commands using a risk scoring matrix as well as blocking known database administrative commands (e.g., DROP, CREATE, etc.). Reports are generated on timestamp, query pattern, reason blocked (e.g., true expression, has "or" token). It has a white list of approved SQL patterns. However, only MySQL database is currently supported. [19] In comparison, the IDPS in this project may be used with any relational database, not just MySQL. The IDPS has both black and white list pattern features.

- *Dot Defender*

It is a web application firewall that offers a SQL-Injection solution. Dot Defender is a multi-platform solution running on Apache and IIS web servers. Central management ensures a single point of control and reporting for all servers. There is an application layer firewall in front of web applications. It has a set of security rules that enable it to be a powerful solution. However, the cost is prohibitive [20].

- *Code Scan Labs' SQL-Injection detection product*

It has the capability to scan web application source code that you selected for code syntax vulnerabilities. It subsequently generates a "debug style" report. The speed depends on how large the web application is and its complexity. The Code Scan software does not fix the code, however; it only points out the issues. The company offers a 21-day free trial, but normally it requires a yearly subscription to be maintained. The actual price is not advertised and one must contact sales representative to find out the cost. A separate activation key is required for different programming languages and additional capabilities [21].

- *Web framework support*

Web frameworks [22] [http://www.php.net/] are used to prevent web applications from the special character [; ' , / @] and meta Characters. But it does not prevent tautologies, illegal, union, piggyback, stored procedures attacks. So attackers can bypass this method.

- *Static analysis*

Static analysis proposed an approach that uses a static analysis can detect and prevent SQLIA in compile time. This technique only focus on tautologies but it cannot detect illegal, union, piggyback, stored procedures SQL injections attacks.

- *Dynamic analysis*

Dynamic analysis is different from Static analysis. However, Static Analysis to detect the queries in compiled time. So the attacker passes advanced queries to web application and it is easily attacked. But Dynamic analysis [23] can used to detect and injection queries prevent in runtime with automatically. But this method do not addresses the issues of illegal, union, piggyback, stored procedures SQL injections attacks.

- *Combined static analysis and dynamic analysis*

The advantage of combined static analysis and dynamic analysis method is to detect SQLIA. It is more complex compared to other techniques. It can solve the problem which cannot be solved by other methods. AMNESIA [13] is a model-based technique that combines the static and dynamic analysis.

6. Design Considerations

The current databases return the results of a statement to the client. The request and results are done by a link from the client to the database back again. The project aims to eliminate the possibility of SQL injection by the use of a filtering proxy server, which will be placed in between the two communicating devices, namely the web application or client and the database. This added layer of protection will allow for the filtering of possible SQL injection attempts and provide the database with a last means of defense. The main step is illustrated in **Figure 1** below.

The main project goals are to:

- Analyze the structure of SQL query commands.
- Build a parser that will check allowable patterns of SQL statements.
- Construct a list of common SQL injection commands.
- Create a proxy server that will alert the database administrator of possible SQL injection commands.
- Prevent a SQL injection attack to a database using this proxy server.
- Prove that SQL injection can be prevented using the filter developed to work on the proxy server.

7. Implementation Decisions

The current paper aims to eliminate the possibility of SQL-Injection by the use of a proxy server, which will be placed in between the two communicating devices. This will allow for the filtering of possible SQL-Injection attempts.

The information flow diagram shows the flow of information between a TDS Proxy server within the domain of this project and the other entities and abstractions with which it communicates. The diagram helps to discover the scope of the system and identify the system boundaries. The system under investigation (TDS Proxy) is represented as a single process interacting with various data and resource flow entities via an interface. As can

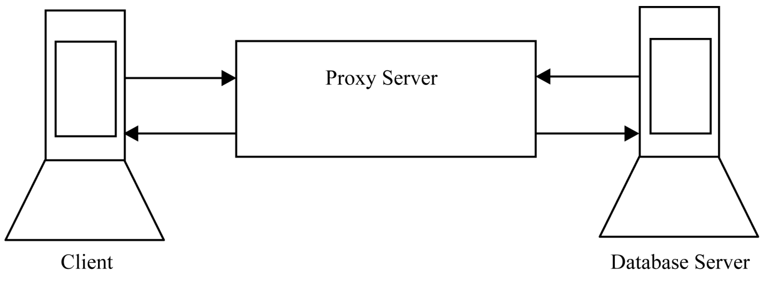

Figure 1. The main implementation step.

be seen from the diagram, the web application provides the query to TDS Proxy which in turn provides safe queries to the database and attack reports to the Database Administrator.

The design and implementation steps made use of the Rational Unified Process (RUP) with the aid of Unified Mark-up Language (UML). This iterative process started off with a simple application and developed into a more complex system in subsequent iterations. The reason for using this methodology was to overcome problem areas in segments. Once the basic concept was conceived and implemented, more advanced featured were added to flesh out the software used for this proof of concept project. The web application is where the queries are formed from the input parameters. These queries are sent to the database through TDS Proxy. The bulk of the system operations take place at the TDS Proxy. When the TDS Proxy has filtered the query, the clean query is sent to the database server. **Figure 2** illustrates how the incoming requests are filtered and only clean queries are passed on to the database for processing. For security reasons, the proxy server will sit on the same machine as the database.

The diagram in **Figure 3** shows all the components in the high level view of the system. The web interface is the tool used by the client to send requests to the database. The web application is pointing to TDS proxy server so that all requests and responses must go through TDS Proxy. The client's web application request triggers the formation of the SQL statement which uses the input parameters of the web form to create the correct SQL statement. This SQL statement is then sent to TDS Proxy. When the SQL statement is received, it is first filtered. Only clean SQL statements are then sent to the database. The database processes the request and sends its response through TDS Proxy. TD SProxy in turn sends the response to the web application for processing to produce the correct view for the client.

The flowchart in **Figure 4** focuses on the internally driven processes as opposed to external events. The action states in the diagram represent the decisions and behavior of the processing. **Figure 4** captures the actions performed at system start-up and run time.

TDS Proxy loads a configuration file at start-up. This file contains, filter settings and options as well as the settings required for the passing of data to the correct destination. Once the system has started, it is able to start receiving data from the client. When data is received from the client, the payload is analyzed. If the payload contains a SQL query, the query is logged and then filtered. If the filter process finds that there is a potential attack, the attack is logged. After logging the attack, the attack information is sent via UDP to the DBA. A false query is sent to the database and the response returned to the client. If the filter process did not pick up an attack, the query is sent to the database and the database response is returned to the client. If the payload does not contain a query, the data is simply passed on to the database.

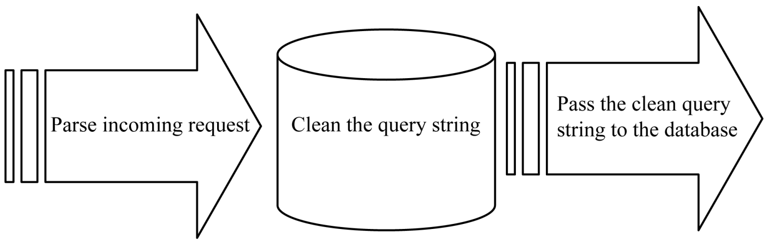

Figure 2. The function of the proxy server.

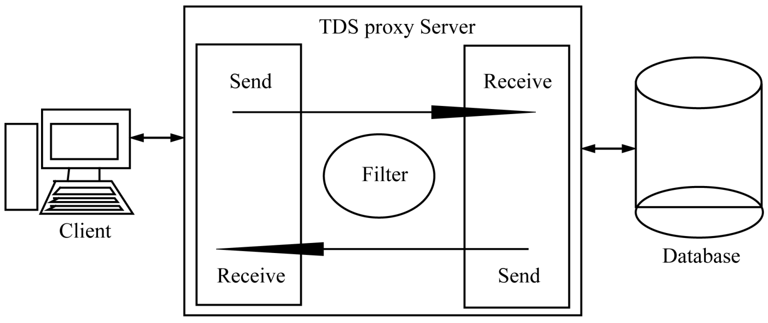

Figure 3. High level design view.

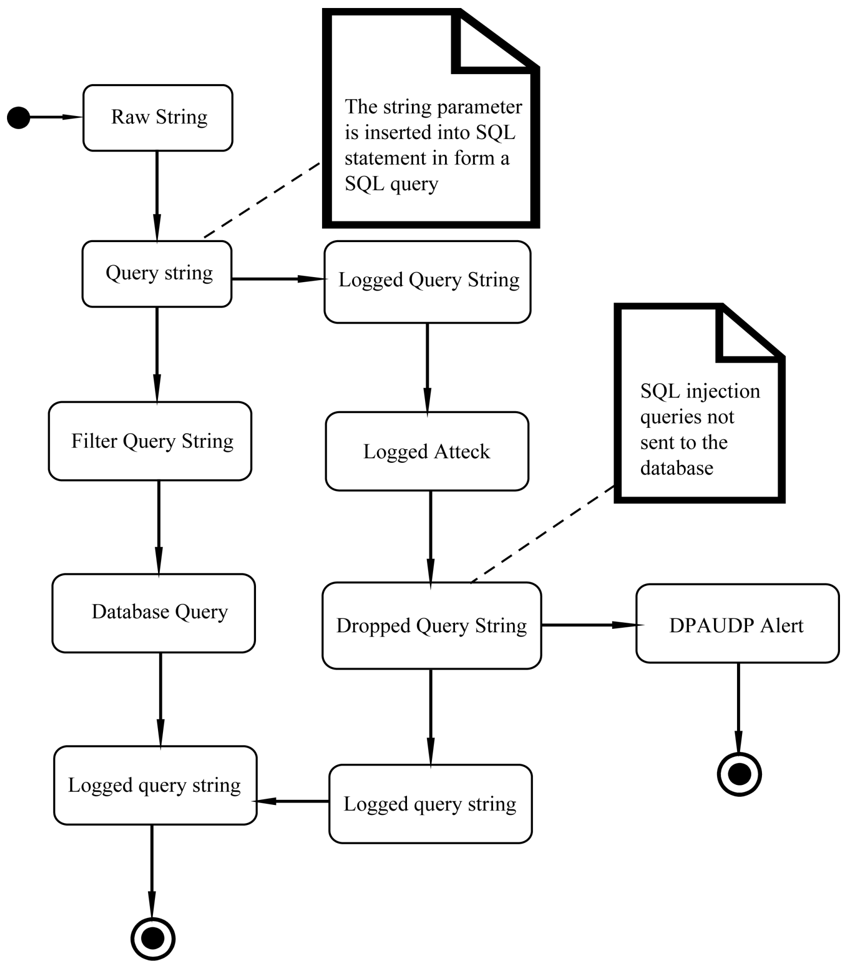

Figure 4. Flowchart of the TDS proxy server.

8. SQL Signature Filtering

The ideal solution is to build a filter that checks for all cases of SQL injection possible. The problem with this is that a list of all possible injection strings is not possible to define [10]. This is suggested by [24] in their paper on "SQL Injection Signatures Evasion". However, going back to the principle of least privilege, by using a white list, it is possible to define what is allowed and thus prevent invalid signatures.

The filtering application should sit as close to the database as possible. Ideally it should sit on the same machine as the database; however, this may have a performance impact due to the filtering process of the filtering proxy server. If the filtering application and the database are on different machines, there is a security risk as the network traffic passes from one machine to the other. With the filtering application sitting on the same machine as the database, there are several advantages.

- There is an additional security as network traffic is limited.
- Processing time is reduced as network latency has no additional effect on the transaction round trip time.
- The filtering application provides a last means of defense for the database.

There are some advantages to using SQL signature filtering as a preventative measure to SQL injection. These are:

- Real time analysis does not impact the database [10].
- Any flaws in the configuration of database privileges or coding of the application would not affect the database security.

However, there are also several disadvantages.

- False positives may also be filtered out in the filtering process [3].

- Packet filtering does not show internal dynamic SQL execution [10].
- This method will not work if the data is encrypted because the strings cannot be viewed in plain text without decryption [4].
- Filtering all incoming http packets may turn out to be resource intensive. A large amount of traffic may need to be handled at the web server [10].

To be a useful intrusion detection system, the filter should be able to find the attackers. Finding the user or attacker means logging login information for inspection. The filter would need a timestamp as well as the source and destination IP address [10].

Most of the suggestions above apply to future deploying of web applications. The (open web application security project) OWASP guide with its many precautions is now becoming an accepted standard [7].

9. Conclusions

There are many vulnerable applications whose code will not be reviewed or patched and it is common knowledge that programmers will continue to produce vulnerable applications. According to [10], there are no commercial solutions to SQL injection. However, several post software packages have been found that claim to prevent SQLIA.

Auditing all of the source code and protecting dynamic input are not trivial. Neither of them is reducing the permissions of all applications users in the database itself. Checking through log files and relying on the least privileges principle do not seem sufficient. Passively detecting SQL injection is not as useful as preventing it in real time. The use of packet sniffers does not allow for the SQL injection prevention as the removal of malicious SQL query statements from the packets is not possible.

This paper has introduced SQL and SQL injection, outlined background research, discussed methods of protecting against SQL injection and presented existing software on the market today. Given the fact that there is a finite set of words in the SQL vocabulary, it seems possible to develop a filter to prevent SQL injection.

References

[1] Anley, C. (2002) Advanced SQL Injection in SQL Server Applications. White Paper, Next Generation Security Software Ltd.

[2] Overstreet, R. (2004) Protecting Yourself from SQL Injection Attacks. http://www.4guysfromrolla.com/webtech/061902-1.shtml.

[3] Imperva Inc. (2004) SQL Injection-Glossary. http://www.imperva.com/application_defense_center/glossary/sql_injection.html

[4] Finnigan, P. (2002) SQL Injection and Oracle. Part One. http://www.securityfocus.com/infocus/1644

[5] Huang, Y., Huang, S., Lin, T. and Tsai, C. (2003) Web Application Security Assessment by Fault Injection and Behavior Monitoring. http://doi.acm.org/10.1145/775152.775174

[6] Microsoft (2003) Secure Multi-Tier Deployment. http://www.microsoft.com/technet/prodtechnol/SQL/2000/maintain/sp3sec03.mspx

[7] Hotchkies, C. (2004) Blind SQL Injection Automation Techniques. http://www.blackhat.com/html/bh-media-archives/bh-archives-2004.html#USA-2004

[8] Microsoft (2003) Checklist: Security Best Practices. http://www.microsoft.com/technet/prodtechnol/SQL/2000/mainain/sp3sec04.mspx

[9] Beyond Security Ltd. (2002) SQL Injection Walkthrough. http://www.securiteam.com/securityreviews/5DP0N1P76E.html

[10] Finnigan, P. (2003) Detecting SQL Injection in Oracle. http://securityfocus.com/infocus/1714

[11] Spett, K. (2002) SPI Dynamics 2005, Inc. SQL Injection: Are Your Web Applications Vulnerable? http://www.spidynamics.com/whitepapers/WhitepaperSQLInject ion.pdf

[12] Grossman, J. (2004) The Challenges of Automated Web Application Scanning. http://www.blackhat.com/presentations/win-usa-04/bh-win-04-grossman/bh-win-04-grossman-up.pdf

[13] Halfond, W.G.J. and Orso, A. (2005) Combining Static Analysis and Runtime Monitoring to Counter SQL Injection Attacks. 3rd International Workshop on Dynamic Analysis.

[14] Imperva Inc. (2005) SecureSphere^{TM}: Dynamic Profiling Firewall^{TM}". http://www.imperva.com/products/securesphere/resources.asp?show=datasheet

[15] Ristic I (2005) "ModSecurity for Java". http://www.modsecurity.org/projects/modsecurity/java/

[16] Seclutions, A.G. (2003) Airlock—Application Security Gateway. http://www.seclutions.com/en/downloads/AirLock_Overview_Nov_2003.pdf

[17] Angelo, C., Corrado, A.V. and Massimiliano, D.P. (2010) A Heuristic-Based Approach for Detecting SQL-Injection Vulnerabilities in Web Applications.

[18] Boyd, S.W. and Keromytis, A.D. (2004) SQLrand: Preventing SQL Injection Attacks. *Proceedings of the* 2*nd Applied Cryptography and Network Security Conference*, Yellow Mountain, 8-11 June 2004, 292-302. http://dx.doi.org/10.1007/978-3-540-24852-1_21

[19] Homepage for GreenSQL. http://www.greensql.net/

[20] About Page for Dot Defender from Applicure. http://www.applicure.com/About_dotDefender

[21] About Page for CodeScan from CodeScan Limited. http://codescan-labs.software.informer.com/

[22] Kosuga, Y., Kernel, K., Hanaoka, M., Hishiyama, M. and Takahama, Y. (2007) Sania: Syntactic and Semantic Analysis for Automated Testing against SQL Injection. *Computer Security Applications Conference*, 107-117.

[23] Parosproxy.org. http://sourceforge.net/projects/dynamicproxy/?source=directory

[24] Maor, O. and Shulman, A. (2004) Blind SQL Injection. http://injection.rulezz.ru/SQLInjectionSignaturesEvasion.pdf