Frameworks for SQL Retrieval on Web Application Security

Haeng Kon Kim

***Abstract*— A SQL injection attack targets interactive web applications that employ database services. Such application accept user input, such as form fields, and then include this input in database requests, typically SQL statements. In SQL injection, the attacker provides user input that results in a different database request than was intended by the application programmer. But, the existing network security techniques are completely inadequate to defend the web application attacks. This paper proposes an P-SQLIAD(Pattern based SQL Injection Attack Detection) to detect the SQL injection attacks using pattern. The P-SQLIAD consists of PCM(Pattern Create Module) and ADM(Attack Detection Module), and is designed to detect an SQL injection of the web application attacks. Using pattern of attacks, the detection time is faster than the existing web application security and reduces the positive false rates.**

Index Terms*— Web Application Security, SQL Injection, OWAPS, Authentication

I. Introduction

The use of the web application has become increasingly popular in our routine activities, such as reading the news, paying bills, and shopping on-line. As the availability of these services grows, we are witnessing an increase in the number and sophistication of attacks that target them.

Specially, HTTP doesn't defend with Firewall, IDS and IPS etc and is open to use Web service, Web application attacks using HTTP are increasing. The Gartner Group states that 75% of the cyber attacks today are at the application level. A staggering 97% of the over 300 Web Sites audited were found vulnerable to web application attack.[1].

In particular, SQL injection, a class of code injection attacks in which specially crafted input string result in illegal queries to a database, has become one of the most serious threats to web application.

Therefore, the existing network security techniques are completely inadequate to defend the web application attacks.

This paper proposes a P-SQLIAD (Pattern based SQL Injection Attack Detection for detecting and preventing SQL injection attack detection. The P-SQLIAD uses pattern based approach to detect illegal queries before they are executed on the database. Also, the detection time is faster than the existing web application security and reduces the positive false rates using pattern of attacks.

Department of Computer Engineering, Catholic University of Daegu
Daegu, Korea 712702
hangkon@cu.ac.kr

The rest of this paper is organized as follows. In section 2 gives an overview of related work and Section 3 and Section 4 present the P-SQLIAD design and evaluation of the implementation, and finally, Section 5 concludes.

II. Related Work

A. Web Application Structure

Web application is the application programs which is used in web browser and consist of three layers such as presentation layer, CGI layer and database layer. Figure 1 shows the structure of web application [2].

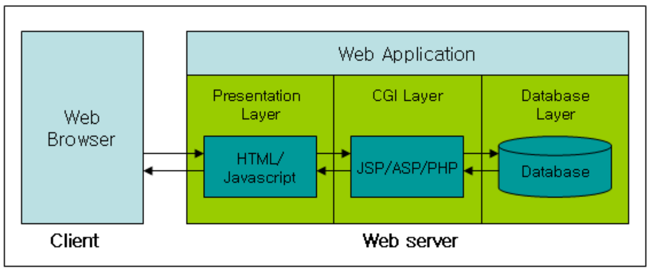

Figure 1: Web application structure

Presentation Layer

The presentation layer is a graphic user interface which collects data from user or shows the result of data process to user. Flash, HTML and JavaScript makes the presentation layer which directly communicates with users.

CGI Layer

The CGI Layer is located in between the presentation layer and database layer. The CGI Layer is called Server-Side Script Process. It translates the data which is inputted from user and saves into database. Also the CGI Layer sends the data of database to the presentation layer.

Namely, the CGI layer substantially processes the data on web application and consists of JSP, PHP, ASP which are Server-Side Scripts Language.

Database Layer

The database layer manages the information which is the result of user data input process and saves into database. It is very important to protect the data from malicious users. Therefore, the database layer permits to access the important data to non-malicious users while denying access to malicious user.

B. SQL Injection Attacks

SQL injection attack is a very old approach but it's still popular among attackers. This technique allows an attacker to retrieve crucial information from a Web server's database. Namely, SQL injection attack is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution.

Example of SQL injection Attack

Figure 2 shows a typical web application in which a user on a client machine can access services provided by an application server and an underlying database. When the user enters a login and a password in the web form and presses the submit button, a URL is generated and sent to the web server.

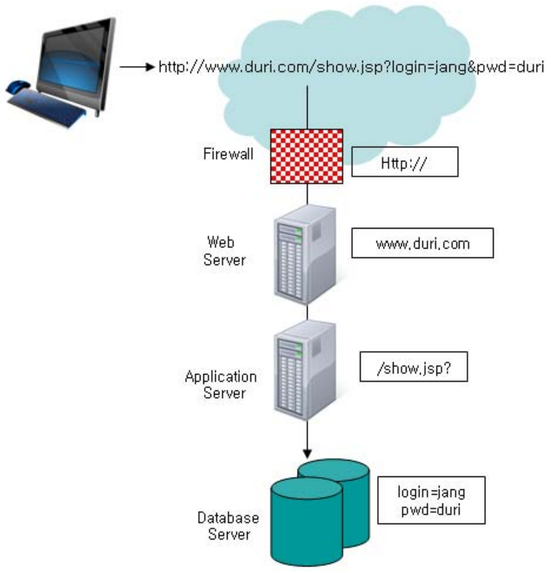

Figure 2: Example of SQL Injection

If both login and pwd(password) are empty, the method submits the following query to the database:

*SELECT * FROM user WHERE login='guest'*

If login and pwd are defined by the user, the method embeds the submitted credentials in the query. Therefore, if a user submits login and pwd as "jang" and "duri", the servlet dynamically built the query:

*SELECT * FROM user WHERE login='jang' AND pwd='duri'*

But, If a user enters "' OR 1=1 -- " and "", instead of "jang" and "duri", the resulting query is

*SELECT * FROM user WHERE login='' OR 1=1 - - ' AND pwd=' '*

The database interprets everything after the WHERE token as a conditional statement, and the inclusion of the "OR 1=1" clause turns this conditional into a tautology. As a result, the database would return information about all users. An attacker could insert a wide range of SQL commands via the exploit [9].

SQL Injection Techniques

a. Tautologies

The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how the results of the query are used within the application. The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an injectable field that is used in a query's WHERE conditional. Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned. In general, for a tautology-based attack to work, an attacker must consider not only the injectable/vulnerable parameters, but also the coding constructs that evaluate the query results. Typically, the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

In this example attack, an attacker submits " ' or 1=1 - - " for the *login* input field (the input submitted for the other fields is irrelevant). The resulting query is:

SELECT accounts FROM users WHERE login='' or 1=1 -- AND pass='' AND pin=

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them [8,11,15,16]

b. UNION Queries

In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. 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.

Referring to the running example, an attacker could inject the text "' UNION SELECT card No. from Credit Cards where acct No.=10032 - -" into the login field, which produces the following query:

SELECT accounts FROM users WHERE login='' UNION SELECT cardNo from CreditCards where acctNo=10032 -- AND pass='' AND pin=

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "CreditCards" table. In this case, the database would return column "cardNo" for account "10032." 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 "cardNo" is displayed along with the account information [8,11,15,16].

c. Piggy-Backed Queries

In this attack type, an attacker tries to inject additional queries into the original query. We distinguish this type from others because, in this case, attackers are not trying to modify the original intended query; instead, they are trying to include new and distinct queries that "piggy-back" on 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. This type of attack can be extremely harmful. If successful, attackers can insert virtually any type of SQL command, including stored procedures,1 into the additional queries and have them executed along with the original query. Vulnerability to this type of attack is often dependent on having a database configuration that allows multiple statements to be contained in a single string.

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

After completing the first query, the database would recognize the 1Stored procedures are routines stored in the database and run by the database engine. These procedures can be either user-defined procedures or procedures provided by the database by default. query delimiter (";") and execute the injected second query. The result of executing the second query would be to drop table users, which would likely destroy valuable information. Other types of queries could insert new users into the database or execute stored procedures. Note that many databases do not require a special character to separate distinct queries, so simply scanning for a query separator is not an effective way to prevent this type of attack [8,12,15,16].

d. Using Comments

SQL supports comments in queries. Most SQL implementations, such as T-SQL and PL/SQL use { { to indicate the start of a comment (although occasionally # is used).

By injecting comment symbols, attackers can truncate SQL queries with little effort. For example, SELECT * FROM users WHERE username='greg' AND password='secret' can be altered to SELECT * FROM users WHERE username='admin' { { AND password=''. By merely supplying admin' {{as the username, the query is truncated, eliminating the password clause of the WHERE condition. Also, because the attacker can truncate the query, the tautology attacks presented earlier can be used without the supplied value being the last part of the query. Thus attackers can create queries such as SELECT * FROM users WHERE username= 'anything' OR 1=1 {{AND password='irrelevant'.

This is guaranteed to log the attacker in as the first record in the users table, often an administrator [11].

C. OWAPS

The primary aim of the OWASP Top 10 is to educate developers, designers, architects, and organizations about the consequences of the most common web application security vulnerabilities. The Top 10 provides basic methods to protect against these vulnerabilities – a great start to your secure coding security program [5].

A1. Cross Site Scripting
A2. Injection Flaws (includes SQL injection)
A3. Malicious File Execution
A4. Insecure Direct Object Reference
A5. Cross Site Request Forgery (CSRF)
A6. Information Leakage and Improper Error Handling
A7. Broken Authentication and Session Management
A8. Insecure Cryptographic Storage
A9. Insecure Communication
A10. Failure to Restrict URL Access

III. P-SQLIAD Design and Implementation

This paper proposes a simple and effective P-SQLIAD(Pattern based SQL Injection Attack Detection) to Detect web application attacks such as SQL injection. P-SQLIAD consists of PCM(Pattern Creation Module) and ADM(Attack Detection Module).

Specially, Whenever PCM parses SQL, PCM creates an SQL_condition and SQL extension pattern , and saves in database. ADM uses the patterns which saved in database to detect SQL injection attract.

Figure 3 shows the process of P-SQLIAD.

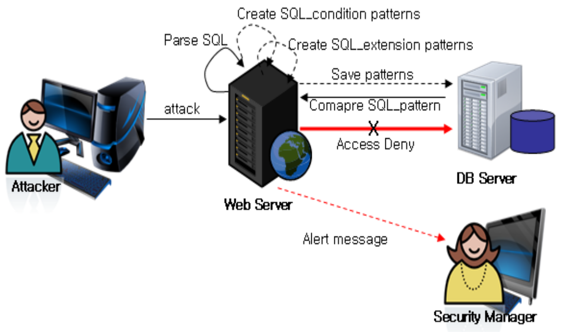

Figure 3: The process of P-SQLIAD

A. PCM Design

PCM(Pattern Creation Module) creates an signature of SQL injection.

The steps of pattern creation are as follows:

Step 1. read SQL_query and parse SQL_query.
Sql_query : SELECT * FROM users WHERE id='jang' AND pwd='duri' -- AND pwd='seri'
The result of parsing :
$\boxed{\text{SELECT}}\ \boxed{*}\ \boxed{\text{FROM}}\ \boxed{\text{users}}\ \boxed{\text{WHERE}}\ \boxed{\text{id}}\ \boxed{=}\ \boxed{\text{'jang'}}\ \boxed{\text{AND}}\ \boxed{\text{pwd}}\ \boxed{=}\ \boxed{\text{'duri'}}\ \boxed{\text{--}}\ \boxed{\text{AND}}\ \boxed{\text{pwd}}\ \boxed{=}\ \boxed{\text{'seri'}}$

Step 2. if condition is existed, SQL_condition pattern is created.
Figure 4 shows the example of SQL_condition Pattern.

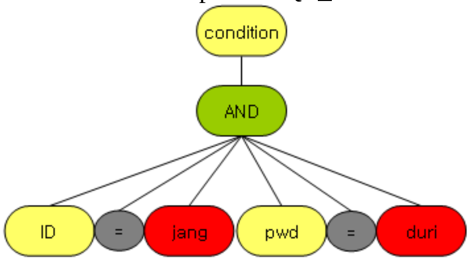

Figure 4: SQL_condition Pattern

Step 3. SQL_conditon pattern is saved.
Step 4. if extension is existed, SQL_extension pattern is created.

Figure 5 shows the example of SQL_extionsion Pattern.

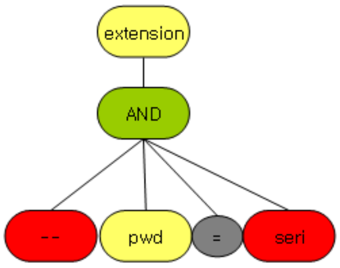

Figure 5 : SQL_extionsion Pattern

Step 5. SQL_extension pattern is saved.

B. ADM Design

ADM(Attack Detection Module) detects an attack of web application using pattern which makes by PCM.

The steps of attack detection are as follows.

Step 1. Read SQL query

Step 2. Compare an pattern of SQL query and the sql_extension of warning table(EXT_t) if the extension exists in query.

Step 3. If the result of Step2 is TRUE, P-SQLIAD calls the manager of Web Server.

Step 4.Compare an pattern of SQL query and the sql_condition of warning table(CON_t) if the condition exists in query.

Step 5. If the result of step 4 is TRUE, P-SQLIAD calls the manager of Web Server.

Figure 6 shows the process of ADM.

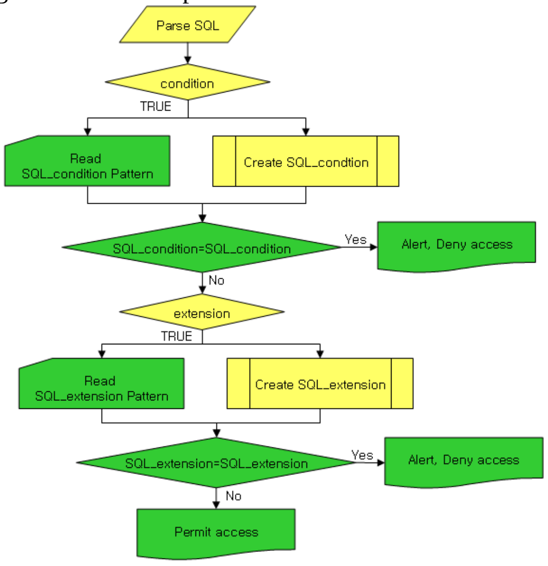

Figure 6 : The process of ADM

IV. Simulation of P-SQLIAD

The simulation environment of the P-SQLIAD proposed in this paper is Intel Pentium 4 CPU 3.00, 2GB RAM, MS-Windows XP Professional OS, and PHP, JavaScript, MySQL.

To evaluate our implementation, we selected three real-world web applications that have been used for previous evaluations in the literature[6,7,8,9]. Each of these web application is provided in multiple web-programming languages, so we used the PHP version to evaluate the applicability of our implementation.

Table 1 shows for each subject the number of attacks and the number of attacks detected by P-SQLIAD. As the table shows, P-SQLIAD achieved a perfect score. For all subjects, it was able to correctly identify all attacks as SQL injections, that is, it generated no false positives.

Table 1: the Result of Simulation

Web Application	P-SQLIAD	AMNESIA[9]
	Attack/detection	Attacks/detection
Bookstore	18/18	182/182
Classifieds	20/20	200/200
Events	26/26	260/260
Portal	14/14	140/140

V. Conclusions

The use of web applications has become increasingly popular in our routine activities, such as reading the news, paying bills, and shopping on-line. As the availability of these services grows, web application system incurs many problems about security, and falls away the capacity of system.

This paper proposes an P-SQLIAD (Pattern based SQL Injection Attack Detection) to detect web application attacks using pattern. The P-SQLIAD consists of PCM(Pattern Create Module) and ADM(Attack Detection Module), and is designed to detect the SQL injection and Cross Site Scripting(XSS) of web application attacks.

The evaluation of P-SQLIAD was used three real applications. The result of the study show that P-SQLIAD was able to stop all of the attempted attacks without any false positives.

Acknowledgment

This work was supported by the Korea National Research Foundation (NRF) granted funded by the Korea Government (MEST No. R- 2009-0083879).

References

[1] http://www.j2ktechnology.com/products_services_security/application_test.html
[2] In-yong Lee, Jae-ik Cho, Kyu-hyung Cho, Jong-sub Moon, “A Method for SQL Injection Attack Detection using the Removal of SQL”, vol.10, no.5, pp.135-147, 2008
[3] The Open Web Application Security Project, “OWASP Top10project” http://www.owasp.org/index.php/Cross-cite_Scripting_(XSS)
[4] http://www.crosssitescripting.com/
[5] The Open Web Application Security Project, “OWASP Top10 Project” http://www.owasp/org/index.php/Top_1-_2007..
[6] GotoCode, http://www.gotocode.com/
[7] Z. Su, G.Wassermann, “The Essence of Command Injection Attacks in Web Applications”, In Conference Record of the 33rd ACM SIGPLANSIGACT Symposium on Principles of Programming Languages, pp.372- 382, 2006
[8] W.G.Halfond, J.Viegas, A.Orso, “A Classification of SQL-Injection Attacks an Countermeasures”, In Proceeding on International Symposium On Secure Software Engineering Ralegh, NC, USA, pp.65-81, 2006.
[9] W.G.Halfond and A.Orso, “AMNESIA : Analysis and Monitoring for Neutralizing SQL-Injection Attacks”, In Proceedings of 20th ACM International Conference on Automated Software Engineering(ASE), pp.174-183, 2005

[10] F.Valeur, D.Mutz, G.Vigna, "A Learning Based Approach to the Detection of SQL Attacks", In proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment, pp123-140, 2005

[11] Gregory T. Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks", In Proceedings of the 5th international Workshop on Software Engineering and Middleware(SEM), pp.105- 113, 2005.

[12] Chris Anley, "Advanced SQL Injection In SQL Server Applications", An NGSSoftware Insight Security Research(NISR) Publication, 2002. URL:http://www.nextgenss.com/papaers/ advanced_sql_injection.pdf

[13] Stephen Thomas, Laurie Williams, "Using Automated Fix Generation to Secure SQL Statements", In proceeding of the 29th International Conference on Software Engineering Workshops(ICSE), pp.54-60, 2007

[14] Z. su, G. Wassermann, "The Essence of Command Injection Attacks in Web Applications", 33rd ACM SIGPLAN- SIGACT Symposium on Principles of Programming Languages, pp.372-382, 2006

[15] S. McDonald, "SQL Injection:Modes of attack, defence, and why it matters", White paper, GovernmentSecurity.org, April 2002. http://www.governmentsecurity.org/articles/SQLInjecetionModesofAttackDefenceandWhyItMatter.php

[16] S.Labs, SQL Injection White paper, SPI Dynamics, Inc., 2002, http://www.spidynamics. com/assets/documents/whitepaperSQLInjection.pdf