SIDP-SQL Injection Detector and Preventer

Saurabh Doshi
Student
flat no. 3 Aster Avenue near
Dilli Dairy Marketyard,Pune.

Chaitali Parekh
Student
Parekh Food Products
At post birwadi Mahad

Ashwini Padale
Student
Shri datta aprts,ft No:3
Katraj Pune 46.

Devata Anekar
Asst Professor
Sinhgad Academy of Engg
Kondhwa Pune.

ABSTRACT

Internet is a very crucial part of today's life. And when we discuss about internet , Web Applications come into focus. Now a days many Web Applications use RDBMS & Web Applications allows its valid users to deal with data stored in RDBMS.

Traditionally mostly programmers have been trained in terms of writing code to implement the intended functionality but they are not aware of security aspect in many ways. The Web Applications are vulnerable to different types of attacks. One of the most dangerous attack is SQL Injection attack.SQL injection is an attack method used by hackers to

retrieve, manipulate, or delete information in organizations' relational databases through web applications. Our technique is implemented in tool named SQL Injection Detector and Preventer(SIDP) which secures Web Applications from different attacks. A great comparative study is made between SIDP and other similar tools and a conclusion is drawn that SIDP is the most efficient tool of all others.

Keywords

SQL – Structure Query Language, SQLIA –SQL injection attack, Positive Tainting, Taint Propagation, Syntax Aware Evaluation, Hard-coded strings,Implicity Created Strings, False Positives, Negative Tainting.

1. INTRODUCTION

Web application is pillar for the Internet & Internet is need of today's world.web applications are popular due to their convenience , flexibility &availability. Everything is vulnerable to attacks. Similarly Web Application requires security from internet thieves/thefts.

Database are fundamentals of Web Applications. Database enable Web Application to store data, preferences and content element using SQL, Web Applications interact with databases to dynamically build customized data views for each user. Here in this tool we use SQL(**S**tructured **Q**uery **L**anguage) for our database design. Database is crucial part of any Web Application and security is the need of it .

There are variant types of attack to which Web Applications are vulnerable some of them are:-

1. Remote code execution
2. Format string vulnerabilities
3. Cross Site Scripting (XSS)
4. Username enumeration
5. SQL injection

1.1 Remote code execution

As the name suggests, this vulnerability allows an attacker to run arbitrary, system level code on the vulnerable server and retrieve any desired information contained therein. Improper coding errors lead to this vulnerability.

1.2 Format String Vulnerabilities

This vulnerability results from the use of unfiltered user input as the format string parameter in certain Perl or C functions that perform formatting, such as C's printf().

1.3 Cross Site Scripting

The success of this attack requires the victim to execute a malicious URL which may be crafted in such a manner to appear to be legitimate at first look. When visiting such a crafted URL, an attacker can effectively execute something malicious in the victim's browser. Some malicious Javascript, for example, will be run in the context of the web site which possesses the XSS bug.

1.4 Username enumeration

Username enumeration is a type of attack where the backend validation script tells the attacker if the supplied username is correct or not. Exploiting this vulnerability helps the attacker to experiment with different usernames and determine valid ones with the help of these different error messages.

In this paper we mainly focus on SQL injection attack as it is increasingly frequent and pose very serious security risks because they can give attackers, unrestricted access to database that underlie Web Applications.SQL Injection is technique used to take advantage of nonvalidated input vulnerabilities to pass SQL a Web Application for execution by a backend database instead of attacking instances such as Web Servers or Operating Systems. The purpose of SQL injection is to attack RDBMS, running as back-end systems to Web Servers, through Web Applications.

2. SQL INJECTION ATTACKS

In general SQL Injection Attack are a class of code injection attacks that take advantage of the lack of validation of user input. There are four main categories of SQL Injection attacks against databases

2.1 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.

2.2 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.

2.3 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.

2.4 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

3. DETECTION OF SQL INJECTION VULNERABILITY

Detection of SQL injection is tough because it may be present in any of the many interfaces application exposes to the user and it may not be readily detectable. Therefore identifying and fixing this vulnerability effectively warrants checking each and every input that application accepts from the user.

How to find if the application is vulnerable or not As mentioned before web applications commonly use RDBMS to store the information. The information in RDBMS is stored/retrieved with the help of SQL statements. Common mistake made by developers is to use, user supplied information in the 'Where' clause of the SQL statement while retrieving the information. Thus by modifying the 'Where' clause by additional conditions to the 'Where' clause; entire SQL statement can be modified. The successful attempt to achieve this can be verified by looking at the output generated by the DB server. Following Example of 'Where' clause modification would explain this further.

If the URL of a web page is:

http://www.prey.com/sample.jsp?param1=9 The SQL statement the web application would use to retrieve the information from the database may look like this: SELECT column1, column2 FROM Table1 WHERE param1 = 9 After executing this query the database would return data in columns1 and column2 for the rows which satisfy the condition param1 = 9. This data is processed by the server side code like servlets etc and an HTML document is generated to display the information.

To test the vulnerability of the web application, the attacker may modify the 'Where' clause by modifying the user inputs in the URL as follows. http://www.prey.com/sample.jsp?param1=9 AND 1=1 And if the database server executes the following query: SELECT coulmn1, column2 FROM Table1 WHERE param1 = 9 AND 1=1 . If this query also returns the same information as before, then the application is susceptible to SQL injection.

In reality , there is a wide variety of complex and sophisticated SQL exploits available to attackers. We next discuss the main types of such attacks.

4. MAIN TYPES OF SQL INJECTION ATTACKS

The main variants of SQL Injection are described in this section and the intent of these attack is also described. The different types of attack are generally not performed in combination; many of them are used together or sequentially depending on the goal of attacker.

4.1 Tautologies:

The main goal of a tautology-based attack is to inject code in conditional statements so that they always evaluate to true. Although the results of this type of attack are application specific, the
most common uses are bypassing authentication pages and extracting data. Here Attacker exploits a vulnerable input field that is used in the query's WHERE Clause.
This attack type has three main goals:

1. bypass authentication
2. identify injectable parameters
3. extract data

An example of this attack is as follows:

SELECT 1234 accounts FROM1234 users WHERE 1234 login=' ' OR 1=1 - - AND1234 pass='' AND 1234 pin= ;

The execution of Query is carried out when output the given condition is true. In this example, an attacker has injected a conditional (OR 1=1) that transforms the entire WHERE clause into a tautology and so every row in the users table will be returned.

4.2 Union Queries:

Tautology is not so power full attack hence attack with help Union Queries came into focus. For example, assume there is another table named creditcards in the same schema as the users table. In that case, an attacker could construct a query like:

**SELECT accounts FROM users WHERE login =' '
UNION
SELECT cardno FROM creditcards WHERE accountno=5050 - - AND pass =' ' AND pin=;**

The first Query returns NULL result and followed Query returns data from "creditcards" table if the given "account no." exists. The outcome of this attack is that database returns a dataset that is union of results of original query with the results of injected query.

4.3 Piggy Backed Queries:

This attack is same as Union Query just here the keyword Union is replaced by a delimiter. The goals of this attack type are:

1. extract data
2. add or modify data
3. perform denial of service
4. execute remote commands

The example is as follows:

SELECT products FROM customers WHERE login='chaitali' AND pass = ' '; DROP TABLE customers ; - - ' AND pin = 1991;

The database treats this query string as two queries separated by the query delimiter (";") and executes both. The second malicious query causes the database to drop the customers table in the database, which would have the catastrophic consequence of deleting all customer information. The above attack has the DROP TABLE statement piggy-backed onto the original query.

5. OUR APPROACH

Till today most of the tools developed for protecting Web Application from SQL Injection attack use/followed Traditional Dynamic Tainting.Traditional Dynamic Tainting identifed untrusted String as tainted ,track the flow of tainted data at runtime,and prevent this data from being used in potentially disaster ways.To make changes in previous approach and develop a new improved tool which followed Dyanamic Tainting.
Unlike existing dynamic tainting techniques, our approach is based on the novel concept of positive tainting, that is, the identification and marking of trusted, instead of untrusted, data. Second, our approach performs accurate and efficient taint propagation by precisely tracking trust markings at the character level. Third, it performs syntax-aware evaluation of query strings before they are sent to the database and blocks all queries whose non literal parts (that is, SQL keywords and operators) contain one or more characters without trust markings.

5.1 Positive Tainting

Positive Tainting is based on identification of the trusted data rather than untrusted data.Traditional Tainting (negative tainting)follows the identification of untrusted data and here positive and negative tainting differ. This conceptual difference has significant implications for the effectiveness of our approach, in that it helps address problems caused by incompleteness in the identification of relevant data to be marked. In the case of negative tainting, incompleteness leads to trusting data that should not be trusted and, ultimately, to false negatives. Incompleteness leaves the Web Application vulnerable to SQL injection attacks . In negative tainting detection of attacks is very difficult . Hence we use positive tainting in our approach. Identifying trusted data in Web Application is often straight forward and always less error prone.

5.2 Accurate and Efficent Taint Propogation

Taint Propogation is carried at runtime.It consists of identifying taint markings assosiated with data,while the data is used and manipulated by users at runtime. Taint Propogation need to be carried out accurately otherwise it would cause the daa to be misused. Our approach consists of:
1)identifying taint markings at correct level of granularity
2)precisely accounting for the affect of functions that operate on the tainted data.
The data consists of charcters.Hence to achive accuracy tainting at charactter level is carried in our approach. Here Strings are constantly broken into substrings for building SQL quries.
Tainting can be carried out even at bit level using bitwise operators.Bit level tainting is more secured but complex to implement and deploy.

5.3 Syntax Aware Evaluation

Postive tainting helps us to create taint markings during execution but for achieveing more security we must be able to use the taint markings to distinguish legitimate from malicious queries.
The key feature of Syntax aware evaluation is that it considers the context in which trusted and untrusted data is to make sure that all parts of query other than string or numerical ,literals consists only of trusted charaters .Before the String is sent to the database for execution syntax aware evaluation of a query string is performed.We use SQL parser for creating tokens of the String.The tokens correspond to keywords,operators & literals.The technique that iterates through the tokens & checks whether the tokens other than literals contain only trusted data. If all tokens are identified aas trusted the query is identified as safe and passed further,if an attack is detected a developer sepecfied action can be invoked.
The developers provide with the external data sources which sholud be trusted and our technique would mark and treat data coming from this sources accordingly.

6.OUR PROPOSED ARCHITECTURE

To evaluate our approach we developed a prototype tool called SIDP that is written in Java `based Web applications. As java is most
commonly used languagage for web application development. Figure above shows high leve architecture for SIDP.It shows a tool developer that developes tool and a web appliaction developer .The developer provides trusted policies and trusted sources which conisits of keyword operators and literals.Our tool SIDP has three main modules which are named as TOKEN CHECKER,STRING LIBRARY and STRING DETECTOR. When the user requests fro a web application a query is fromed which is then passed to our tool. If
the query is identifed as trusted the data is retrived from database else developer specified action is carried out.

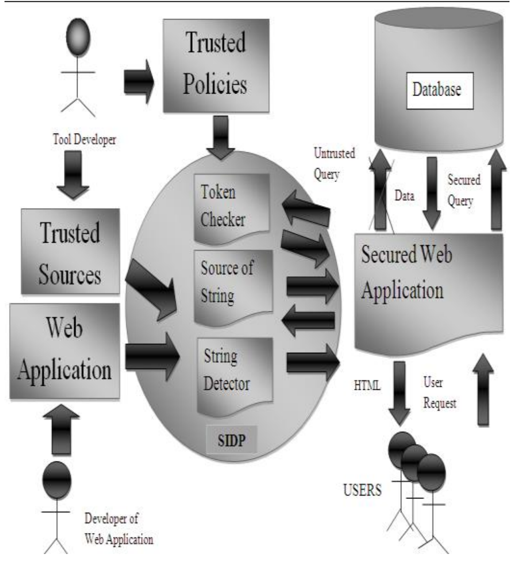

Fig.1 Architecture of SIDP Tool.

6.1 String Library

String is our library of classes that mimic and extend the behaviour of java's standard string classes . The library takes advantage of the object oriented features of java language to provide complete mediation of string operations that could affect string values and their associated trust markings.

7.ANALYSIS AND DESIGN

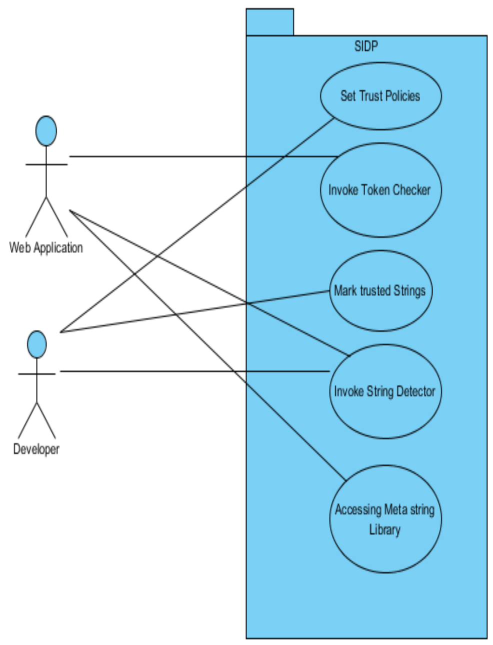

Fig 2. Context Level Use Case Diagram .

This diagram is Use Case Diagram for tool SIDP.A Use Case model represents the Use Case view of the system. This view is important, as it effects all other views of the system. Both the logical and physical architecture are influenced by the use cases. The above diagram consists of 2 actors and SIDP system consisting of different use cases. The actors are the Developer and the Web Application which are the inputs to the system. The Web Application invokes a token checker, String Detector and access the Meta String library. Similarly Developer Sets the Trust policies and marks trusted String.

The next figure represents the Sequence diagram for tool SIDP. The Sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur. Sequence diagram are useful in documenting how a future system behave. The user communicates with the database through Web Application ,controller and the library table.

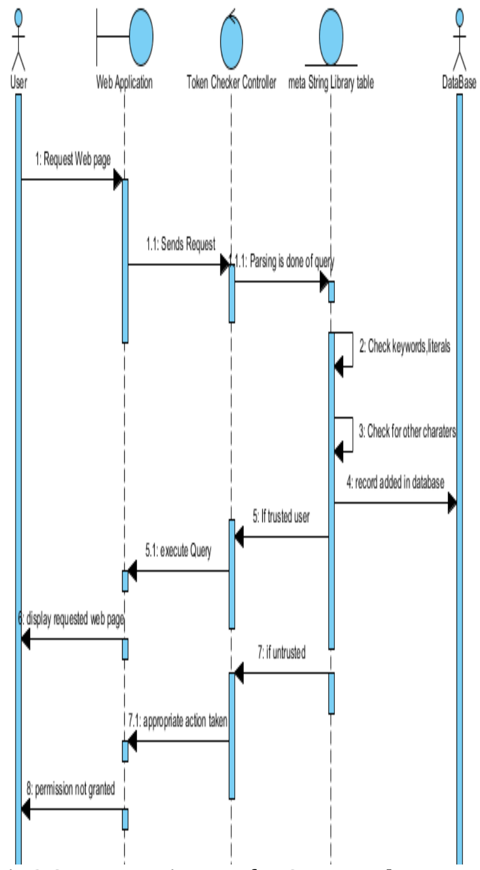
User
Web Application
Token Checker Controller
meta String Library table
DataBase
1: Request Web page
1.1: Sends Request
1.1.1: Parsing is done of query
2: Check keywords,literals
3: Check for other charaters
4: record added in database
5: If trusted user
5.1: execute Query
6: display requested web page
7: if untrusted
7.1: appropriate action taken
8: permission not granted

Fig.3.Sequence Diagram for SIDP Tool.

8. CONCLUSION

This tool presented a novel highly automated approach for protecting Web applications from SQLIAs. Our approach consists of 1) identifying trusted data sources and marking data coming from these sources as trusted, 2) using dynamic tainting to track trusted data at runtime, and 3) allowing only trusted data to form the semantically relevant parts of queries such as SQL keywords and operators.

Our approach also provides practical advantages over the many existing techniques whose application requires customized and complex runtime environments: It is defined at the application level, requires no modification of the runtime system, and imposes a low execution overhead.

We have evaluated our approach by developing a prototype tool, SIDP, and using the tool to protect many applications when subjected to a large and varied set of attacks and legitimate accesses. SIDP successfully and efficiently stopped over 12,000 attacks without generating any false positives

9. ACKNOWLEDGMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this paper. I want to thank Halfond , W.G.J, A.Orso for their guidance through their papers related to SQL injection.

10. REFERENCES

[1] Halfond, W. G. J. and A. Orso (2005). Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks. Workshop on Dynamic Analysis (WODA 2005). St. Louis, MO,USA, ACM**:** pp. 1 - 7.

[2] Shaukat Ali, Azhar Rauf, Huma Javed:SQLIPA: An Authentication Mechanism Against SQL Injection.

[3] Top ten most critical web application vulnerabilities, 2005.

http://www.owasp.org/documentation/topten.html

[4]William G.J. Halfond, Alessandro Orso, and Panagiotis Manolios: Using Positive Tainting and Syntax Aware Evaluation to Counter SQL Injection Attacks.

[5]W.G. Halfond and A. Orso(2005) 'AMNESIA: Analysis and Monitoring for NEutralizing SQLInjection Attacks', In the Proceedings of 20th IEEE and ACM International Conference onAutomated Software Engineering, pp. 174-183.