

Prevention of SQL Injection

CSE 545: Software Security
Professor: Dr. Adam Doupe

Aakash Chaurasia
Computer Science (MCS)
achauras@asu.edu

Abstract

Software Security has become an integral part of the system. With the increasing day by day demand of applications, threats and vulnerabilities are generating and propagating rapidly. To address this issue Software Security has become an important aspect of each application. With the rapid generation of data and databases, attacks on database are becoming more common. Attackers attack database, which puts whole system to vulnerable state. One of this attack is SQL Injection attack. This report discusses the threats posed by SQL Injection attacks and preventive measures to avoid attacks from happening at first place.

Keywords

Software, Application, Threats, Vulnerability, Database, SQL Injection, First Order, Second Order, Lateral Injection, Input Sanitization, GET, POST, SQL Query, SQL Syntax.

Introduction

With day by day increasing need of software and ease of human being, new designs and architectures are being proposed. Since no design is perfect, each design and architecture has its flaws. There is a black hat world available outside who are continuously looking for such flaws, exploit it for their benefits [1]. It's a major concern to handle these attacks and try to prevent them. That's the scenario when Software Security is used. It's a concept of an engineering software, which is able to run properly and with its full extent, without breaking down under any attack [1]. Most of the programmers had started taking this issue seriously, but very little number has knowledge to avoid attacks [1]. Its main cause is software bug which turns to defect, which turns to error, which leads to failure [1]. Among various attacks possible there is one famous attack possible on database named SQL Injection Attack. This kind of attack comprises of insertion of "injection" of SQL query to the database or to the system using the input boxes which are available on the front end [5]. These vulnerabilities are generated when there is a bug in the code. A SQL injection attack can consist of many possible scenarios like reading from the database, writing to the database, executing the administrative privileges, etc. There are different types of attacks possible in SQL Injection like first Order Second Order attacks.

Project Description & Implementation

In this project we are going to learn and study SQL Injection and different types of attacks possible in detailed level. Since SQL injection demonstrates the maximum threat to the application, it needs to be handled [2]. It authorizes attackers to get the unrightfully access to the databases, which has the sensitive information, which should be kept secure [2]. SQL Injection poses threats to all the four Security principles Confidentiality, Integrity, Availability, and Authorization [5]. SQL Injection can further be divided into three parts First Order, Second Order, Lateral Order, varying from simple select attack to executing high level commands attack [6].

First Order

In this kind of attack attacker always input data such a way that existing sql query is modified and its intended purpose to perform is changed. With such kind of attacks sensitive data is exposed to the attacker. There are three sub categories in this

UNION

Attacker can just add UNION clause at the end of the query by inputting in input box. Using this attacker can retrieve whichever information or table he wants. By SQL definition it is allowed to append results from multiple tables unless they all have same number of columns retrieved [3]. Consider a basic query which displays the single column from database like `Select col1 from user where id = 'username' and password = 'secret'`, where username and password comes from input form, so in this case if instead of secret attacker type `"secret" UNION SELECT user_name FROM USERS"`. So it will display all the users available in the system [3].

SubQuery

In this attacker writes the whole query after input parameter starting with semi colon. For example if query is `Select 1 from user where userid = 10001` in this instead of 10001 user will type `"10001; Select password from user;"`. So in this case SQL Engine will first execute the query and return 1. Immediately after that it displays all the passwords.

Short-Circuit

In this attacker always appends OR short circuit expression with # to the existing query. For example in previous query attacker will input `"XYZ" OR 1 = 1 --"`. So query will

become “Select 1 from users where username = ‘XYZ’ OR 1 = 1 – and password = ‘ ’”. So it will ignore the password where condition and bypass it.

Second Order

Unlike first order Attacker never use input data boxes to attack the application, instead one just stored the malicious data inside the database, which can be further used to process the data and get the result [6]. For example consider a query “Select ssn from employee where name = ‘XYZ’ and password = ‘secret’ ”. In this case attacker will already insert the data in table with name = “XYZ’ OR USERNAME IN (‘ALICE’, ‘BOB’, ‘CATEY’) --”. So whenever attacker use this query it becomes a legitimate query and returns the SSN of all the employees.

Lateral Injection

This kind of attack helps attacker to use stored procedures as a vulnerability. Stored procedure returns a value even if it is not required to [6]. Usually procedures are set of operations which executes something [4]. This procedures are highly vulnerable because they are available to the server through different channels, which connect to client program [4]. This attacks helps to gain information through PLSQL procedure. In this already available function like to_char() can be modified simply by using NLS_Date_Format or NLS_Numeric_Characters [6].

Implementation

We have created a PHP library which serves input sanitization. This library can be imported by the programmer can be used to take input parameters from the form. By sanitizing it means we perform several types of check which needs to be executed before sending input parameters to the database.

For this we have setup the phpmyadmin as the localhost. In this localhost we defined a simple http page which has two options. First option is to access the file for the following user, second is to authenticate the user and upload file. Then we created database with two table’s *files* for storing file and *auth_login* for authentication. Then we populated this tables with the dummy values for testing. Once whole framework was setup, we use sql injection query for first which was available on the oracle website [6]. Then we manipulated the database by following the same oracle site this time for second order sql injection [6]. Then we developed a php script named **sanitize_input.php** which can be used and imported in any php scripts and use the function available *getparam()*.

Getparam() takes two input parameter first input parameter type of the parameter, second is the name of the parameter, and it returns the sanitized value which is used in sql query. There are two types of the parameter GET and POST. In GET method input variable is available in the url, in POST input variable is hidden from the user. So for the GET method function looks like getparam(‘get’, ‘param1’) and for POST method function looks like getparam(‘post’, ‘param1’).

Our library works for some predefined patterns, which we handled using different scenario. If there is any match in the

pattern it will return an error or it will return the result. So we identified all the patterns which are currently possible in First Order and Second Order and sanitized it. Algorithm for the getparam function is given below:

```

1 if type == 'get'
1.1 store $_get[param] in $str
2 if type == 'post'
1.2 store $_post[param] in $str
3 call checkParam with $str input param
3.1 create temporary variable $str to hold $str
3.2 if index of ' in $str is 1 continue
3.2.1 replace $str with substring of $str from index to end
3.2.2 remove all the spaces from $str and convert it to uppercase
3.2.3 if $str contains "OR"
3.2.3.1 if index of "=" in $str == 0
3.2.3.1.2 return substring of $str from 0 to index of ""
3.2.3.2 if index of "=" in $str == 1
3.2.3.2.2 return after replacing "" with "" in $str
3.2.4 if $str contains "UNIONSELECT"
3.2.3.1 if index of "FROM" in $str == 0
3.2.3.1.2 return substring of $str from 0 to index of ""
3.2.3.2 if index of "=" in $str == 1
3.2.3.2.2 return after replacing "" with "" in $str
//Similarly it will do for the rest with respective keywords
3.2.5 if $str contains ";INSERT"
3.2.6 if $str contains ";UPDATE"
3.2.7 if $str contains ";DELETE"
3.2.8 if $str contains ";SELECT"
3.2.9 if $str contains ";DROP"
3.2.10 if $str contains ";ALTER"
3.2.11 if $str contains ";RENAME"
3.2.12 if $str contains ";CREATE"
3.2.13 if $str contains ";TRUNCATE"
3.2.14 if none of the above is true goto 3.3
3.3 if index of ' in $str is 0 return $str

```

There are some considerations for our php script.

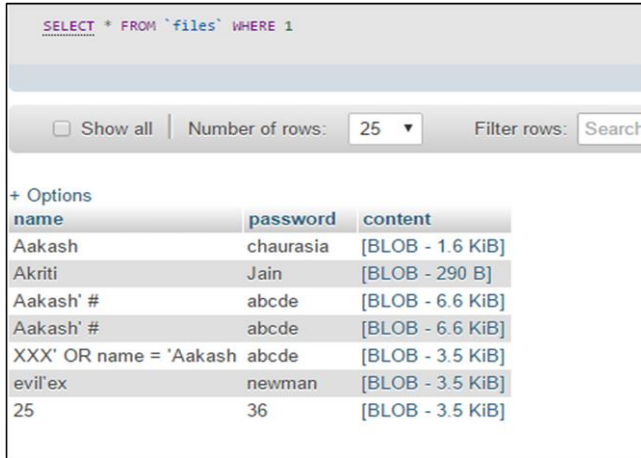
- User has to first import the sanitize_input.php in their php files.
- For inputting data from GET method or POST method user have to use getparam(‘get’, ‘param1’) instead of \$_GET[‘param1’] and \$_POST[‘param1’]
- Using can be cumbersome, if user is taking multiple inputs, one has to call getparam multiple times.
- We are using only those SQL injection patterns which are currently available, in near future it might be possible that, there are more patterns generated, which won’t be handled by the script.
- In php we cannot concatenate function return value to string directly, so user have to store the result in variable first, then append it to the string.
- We are not handling advance SQL injection attacks like Lateral Injection.

Results

We have learned different sql injection attacks. To avoid those attacks we have designed php script, which is just needed to be imported by the programmer. We have handled the first Order attack and Second order attack. We also have test cases for both the attacks with screenshots of before and after scenario.

Database

Our database looks like below it has table named files which stores files for user and we have authentication table for users.

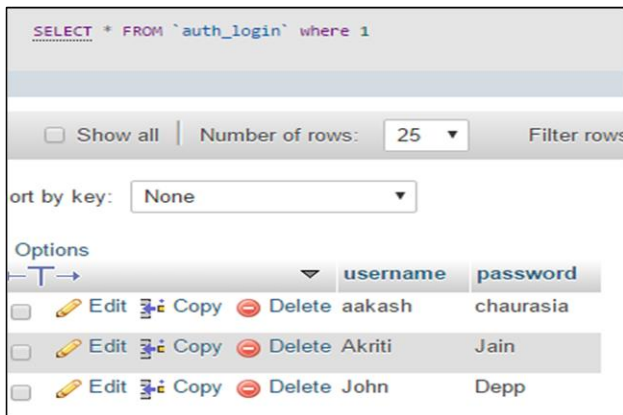


SELECT * FROM `files` WHERE 1

Number of rows: 25

name	password	content
Aakash	chaurasia	[BLOB - 1.6 KiB]
Akriti	Jain	[BLOB - 290 B]
Aakash' #	abcde	[BLOB - 6.6 KiB]
Aakash' #	abcde	[BLOB - 6.6 KiB]
XXX' OR name = 'Aakash	abcde	[BLOB - 3.5 KiB]
evil'ex	newman	[BLOB - 3.5 KiB]
25	36	[BLOB - 3.5 KiB]

Fig 1: Files Table



SELECT * FROM `auth_login` where 1

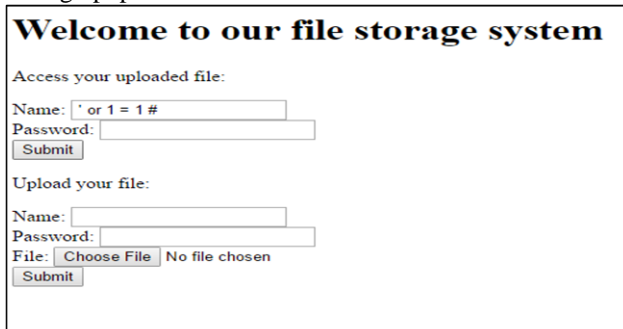
Number of rows: 25

Sort by key: None

username	password
aakash	chaurasia
Akriti	Jain
John	Depp

Fig 2: Login Table

We have designed input form to connect to the database through php.



Welcome to our file storage system

Access your uploaded file:

Name: ' or 1 = 1 #

Password:

Submit

Upload your file:

Name:

Password:

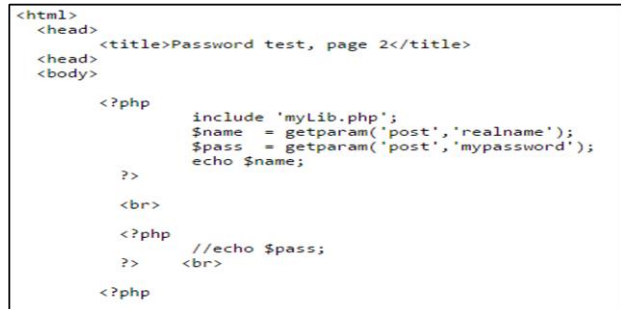
File: Choose File No file chosen

Submit

Fig 3: Input form

First Order Attacks

First SQL Injection query we used was " ' OR 1 = 1 #". This query displays the content of the file and breaks the system as below.



```
<html>
<head>
<title>Password test, page 2</title>
</head>
<body>

<?php
    include 'myLib.php';
    $name = getparam('post','realname');
    $pass = getparam('post','mypassword');
    echo $name;

?>

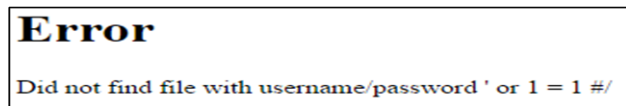
<br>

<?php
    //echo $pass;
?>

<?php
```

Fig 4: Test case before lib for " ' OR 1 = 1 #"

After including the library inside the php script, it throws an error that its an invalid sql query like below.

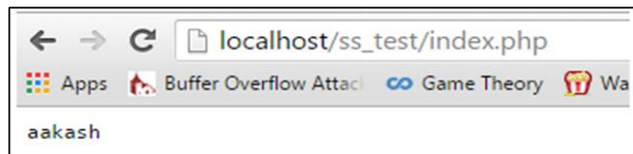


Error

Did not find file with username/password ' or 1 = 1 #/

Fig 5: Test case after lib for " ' OR 1 = 1 #"

Next input parameter was including " ' union select username from auth_login #". This input parameter displays the authorized username from the system.



localhost/ss_test/index.php

Apps Buffer Overflow Attack Game Theory Wa

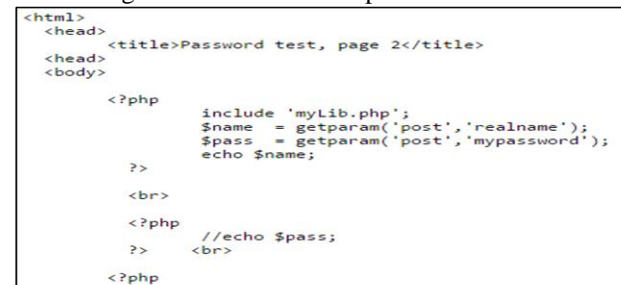
aakash

Fig 6: Test case before lib for Union

After including library Union attacks has been handled and php script throws error showing sql query is incorrect.

Second Order Attacks

As discussed above second order attacks deal with injection value in database and used it afterwards, we have values inserted in files table with name "Aakash' #". This input parameters comments the rest section returns the original file which belongs to Aakash instead of returning "Aakash' #". Below given screenshot is output of this attack.



```
<html>
<head>
<title>Password test, page 2</title>
</head>
<body>

<?php
    include 'myLib.php';
    $name = getparam('post','realname');
    $pass = getparam('post','mypassword');
    echo $name;

?>

<br>

<?php
    //echo $pass;
?>

<?php
```

Fig 7: Test case before lib for Second Order

After incorporating the library in php program, it returns error showing it's an invalid select query.

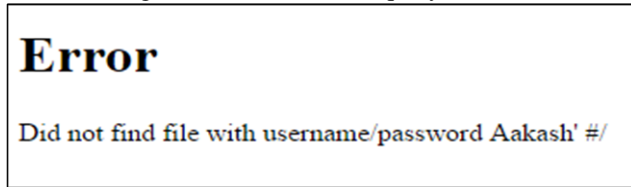


Fig 8: Test case after lib for Second Order Attack

Individual Contribution

Final project was a group project, so we were team of two who worked on this project. We have equally divided our task throughout the phase of the project.

- Walkthrough and research was done by both the project team members.
- I have read paper on first order SQL injection and the detection along with the prevention of the same
- I have created the sql queries for the database and database setup was done by other partner.
- I have written PHP code for exploiting the first order sql injections through webpage.
- Created a php code to secure the first order SQL injection and to sanitize the input logic for first order attack
- Created test cases and performed manual testing on the second order SQL attacks

Learnings from the project

This Coursework and project helped me to get hands on experience and learn following technologies and skill.

- *SQL Injection*
Learned How Sql Injection works, how to find which site is vulnerable and how attackers exploit sql injection vulnerability
- *First Order Injection*
How First Order attack is exploited to get the data from the database and to bypass the authentication security. In our project it was used to exploit and fetch the code saved in file.
- *Second Order Injection*
Perceived hoe second order can be used to attack the database tables by modifying the values in the table, which can be used in further attacks. In our project we inserted record in table to access previously stored value in table
- *Lateral Injection*
Learnt how to attackers exploit lateral injection to modify the sql procedure to take raw input from user.
- *PHP*
Started coding with php, with database connections and hosting as service to be accessed from application
- *Prevention libraries*

Learnt various ways to protect website and to avoid sql injection which is serious threat to database

- *Database*
Started working on SQL and queries like execute Immediate, Union, etc
- *phpMyAdmin*
This was the first webhost project which helped me to understand what needs to be done to deploy a website on phpMyAdmin and php files can be used as a service.

Acknowledgement

I would like to thank Dr. Adam Doupe for his great support throughout the course work and project. For giving us deep understanding of security concepts and making us security expert. Thanks for teaching us theoretical concepts as well as practical implementation using various practical and real time problems which exist real world threat problems. For helping us in the implementation of security in various programs, application and at different levels of implementation and networks. I would also like to thank TA Sai PC, who was very supportive and provided guidance throughout the project and cleared our doubts from basic security concepts to advanced SQL injection prevention implementation, and being available through all the assignments and final project.

Team Members

There were total two team members in this project.

- Akriti Jain
akriti.jain@asu.edu

References

- [1] McGraw, Gary. "Software security." IEEE Security & Privacy 2.2 (2004): 80-83.
- [2] Halfond, William G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures." Proceedings of the IEEE International Symposium on Secure Software Engineering. Vol. 1. IEEE, 2006.
- [3] Buehrer, Gregory, Bruce W. Weide, and Paolo AG Sivilotti. "Using parse tree validation to prevent SQL injection attacks." Proceedings of the 5th international workshop on Software engineering and middleware. ACM, 2005.
- [4] Wei, Kei, Muthusrinivasan Muthuprasanna, and Suraj Kothari. "Preventing SQL injection attacks in stored procedures." Software Engineering Conference, 2006. Australian. IEEE, 2006.
- [5] https://www.owasp.org/index.php/SQL_Injection
- [6] http://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01_tm_attacks.htm