|  |
| --- |
|  |
| Capstone ExperienceIST 894 |
| Lab 8 – Web Application Security: SQL InjectionScott Finlon |

Table of Contents

[1. General Context 3](#_Toc86485131)

[Technical Context 5](#_Toc86485132)

[2. Solution 6](#_Toc86485133)

[References 11](#_Toc86485134)

# General Context

In this lab we look at SQL injection attacks. SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database (*What Is SQL Injection?*, n.d.). SQL injection attacks allow an attacker to view data that they are otherwise unauthorized to see. SQL injection attacks can target applications themselves, the underlying databases, and even the host operation systems in some cases.

SQL injection attacks are performed by injecting SQL queries inside a input field that is otherwise not validating or sanitizing the input. By injecting outside queries an attacker is able to retrieve hidden data, subvert application logic, gain information about the database, retrieve information from other database tables, or just send the host machine commands to run without any feedback. For an example of how this is executed, we can take a look at this PHP code:

$query = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

A form field will expect a value for ID, which might be the number 1, however if someone entered 1’ OR ‘1’ = ‘1’ # it tells the logic that the ID is 1 or if 1 equals 1 which is always true. Since it’s always true, it returns all the values in the databse.

According to OWASP (*A03 Injection - OWASP Top 10:2021*, n.d.), applications are vulnerable to injection attacks when:

* User-supplied data is not validated, filtered, or sanitized by the application.
* Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
* Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
* Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

So to secure this vulnerability, first validate and sanitize the input, but also parameterize the query. Another thing you can do that won't necessarily harden the system but will slow down attackers is to set a limit on the number of results that can be returned. If you expect that a query should only ever return 1 response, set a limit so that it's not possible to return more than 1. This way even if an attacker finds an injection point, it might slow them down enough that you can identify them and stop them before they get everything in the database.

# Technical Context

Injection attacks were the number one entry on the OWASP Top 10 for a number of years, in 2021 it’s slipped down to number three but it’s still a very important web vulnerability. According to OWASP, 94% of the applications that were tested were tested for some form of injection vulnerability and the average incidence rate was 3% with a max incidence rate of 19% (A03 Injection - OWASP Top 10:2021, n.d.).

Injection attacks have remained a top threat for so long because of their versatility. I explained briefly how you can pass a statement that will always be true like 1=1, but injection attacks can also can change application logic. For example, if a website is meant to be passed “if user=’admin’ and password=’password’”, but an attacker inputs “admin’ # “ as the username it completes the quote string and then passes a comment tag which tells the program to ignore the rest of the line of code. In this instance the password is no longer relevant or checked, as long as the username=admin it returns true and grants the attacker access.

There are some caveats to the prevention strategies, as they do have to be implemented correctly in order to be effective (SQL Injection Cheat Sheet, n.d.). While input validation helps filter out injection attempts, the validation needs to happen on the server side. You can’t rely or trust anything that comes from the client, once code is on the client it’s owned by the person at that client. They can do anything they want with it and modify it any way they choose before sending data back, so all trusted validation needs to be on the server. Ensure that the user that is running the web server and communicating with the database has restricted privileges. You want to follow the principle of least privilege and only give the bare minimum that the accounts need in order for the application to function. When attackers attempt to interact with the host operating system and take over a host through injection attacks, they usually take advantage of the connected user having high access rights than it needs which allows the attack to do queries and send system commands that they otherwise shouldn’t be allowed. In the end, following basic security best practices like least privilege, and validating and sanitizing input in a trusted location are the best defenses again injection attacks.

# Solution

To do this lab, we start up the Cyber Basics VM in the US Cyber Range. Once we log in to the machine, we open a web browser and go to http://dvwa.example.com. This is a purposefully vulnerable web application server that allows testing and training to happen more easily. We log into DVWA and click on SQL Injection to attempt this exercise on the ‘low security’ setting.

The idea of this page is to enter a number into the ‘User ID’ field, and then the server will respond with the first and last name of the person with that User ID. However, because the ID field isn’t validated or sanitized for malicious characters, you can modify the query that is placed. If, instead of entering ‘1’, we enter “1’ or user\_id=’2’ # “ the server will respond with any entries for user ID 1 and user ID 2.



Figure - Form as intended



Figure - Vulnerable query



Figure - Injecting additional user query

****

Figure - Sample users table

Figure 4, above, shows what a sample users database table looks like. In SQL, a UNION statement lets you execute one or more additional SELECT queries and append the results to the original query (SQL Injection UNION Attacks | Web Security Academy, n.d.). The caveat here is that all queries need to return the same number of fields. So if we modify the previous query that gave us user 1 and 2 to take out the ‘OR’ statement and put a ‘UNION SELECT’ in it place we can return more information. Entering “’ UNION SELECT user, password from users # “ passes a blank value to the user ID query which returns nothing but then searches for all user/password combinations in the users table and prints them all on screen. Since UNION requires a specific number of fields, sometimes you need to perform different queries to gather more information about the database format and makeup.



Figure - UNION attack to retrieve all passwords



Figure - identifying additional database tables

 The medium security setting of DVWA makes a couple small changes. It gives a drop down list of user ID’s that are able to be queried and it removes the quotes from the query and filters quotes from the field, so injection attacks relying on quotes will no longer work, but UNION attacks are still easily achieved since the drop down list ends up on the client side it’s trivial to edit the HTML right in the web browser to change the value to your injection code and still return all of the users and passwords. These types of password stealing attacks are dangerous because even though these passwords are hashed, they are not secure. The passwords in this database are hashed with MD5 and can be cracked in less than a second due to the lack of complexity they have.



Figure - Modifying the HTML field values



Figure - Successful attack in drop down list



Figure - MD5 Passwords cracked

 The high security version of this form uses a separate window to send the query to the web server, but similar to the low setting it utilizes quotes around the variable again, so one quick modification to add a leading single quote and the UNION SELECT statement returns all of the entries again.



Figure - High Security Successful Injection attack

# ****References****

*A03 Injection—OWASP Top 10:2021*. (n.d.). Retrieved October 29, 2021, from <https://owasp.org/Top10/A03_2021-Injection/>

*SQL injection cheat sheet: 8 best practices to prevent SQL injection | Snyk*. (n.d.). Retrieved October 29, 2021, from <https://snyk.io/blog/sql-injection-cheat-sheet/>

*SQL injection UNION attacks | Web Security Academy*. (n.d.). Retrieved October 29, 2021, from <https://portswigger.net/web-security/sql-injection/union-attacks>

*What is SQL Injection? Tutorial & Examples | Web Security Academy*. (n.d.). Retrieved October 28, 2021, from <https://portswigger.net/web-security/sql-injection>