PSI-CAD-Project Report SoSe 2019

Alexander Böhner (1937944)
`mailto:alexander.boehner@stud.uni-bamberg.de`

Melanie Vogel (1738258)
`mailto:melanie-margot.vogel@stud.uni-bamberg.de`

September 30, 2019

Contents

List of Figures

1 Instructor Guides

1.1 BottleNet

This scenario covers the topic of Blind SQL Injections and is separated into two tasks where the first task shall prepare the attacker for the *real* attack in the second task.

SQL injection refers to all attacks where queries are injected via data input form into a client application. The injection then is executed by predefined SQL statements i.e. dynamically constructs the query when the input is stored in a variable. Whenever this is possible, the attacker can manipulate (INSERT, UPDATE, DELETE) data in the database such as passwords etc. or other administrative tasks. The prerequisite to perform such kind of attacks is that the SQL language is used. However, the interacting platform or database i.e. whether MySQL, sqllite or PostGreSQL does not matter.

1.1.1 Related Work

To gain an understanding on how such attacks work, a complete sql injection attack can be viewed in several tutorials on the web[1][2].

An overview on SQLi attack types and example applications can be found in [H+06], including essential concepts that are useful for this scenario. These essential concepts include the use of tautologies and UNION queries and they are both explained in an understandable way. This paper also includes an evaluation of detection-focused and prevention-focuses techniques/tools from research and the techniques are described according to the attack types.

For example a highly cited detection-focused tool is the monitoring tool AMNESIA [HO05]. Prevention-focus mechanisms such as [BK04] present architectures that do not even make it possible to perform SQLi attacks.

From the web research, several open-source SQL injection tools to perform such attacks exist[3]. For example SQLMap - Automatic SQL Injection And Database Takeover Tool is a commonly known and used tool.

In this scenario we present error-based SQLi and blind SQLi where the first one is often referred to as the *standard* SQL injection since it was used in the early age of

[1]https://www.youtube.com/watch?v=WFFQw01EYHM
[2]https://portswigger.net/web-security/sql-injection/union-attacks
[3]https://kalilinuxtutorials.com/sql-injection/

SQL injections. However, since the attack is one of the most common attacks, website providers learned from the error-based attacks and try to remove the basis for this attacks i.e. the error messages that provide useful information for the attacker. Therefore blind SQL attacks should be investigated and how they can be applied[Hot04].

1.1.2 Task: Warm-Up

This task should help to gain some understanding of general SQL attacks and the difference between blind SQL and error-based SQL injection. Find out which type of SQL injection attack we use here.

The secret is basically the answer whether it is an error-based or blind sql injection attack. The goal of the task is to sensitize attackers for the difference between the attacks.

1.1.2.1 Special Notes

- Wild cards are used in the db query [4].

1.1.2.2 Guide

The task can be guided as follows including additional hints to achieve the goal and understand the basics of the attack:

1. **Which input fields are vulnerable? Start with a basic SQL attack**

   The basic SQL attack starts with trying to provoke an error message. At first, the attacker must detect the vulnerable input field. Some fields use a prepared statement, some not. However, the search field is the only field that is possible. A prepared statement will make a check before executing the query to the database, where a plain hand-over from the input field can modify the query. In order to produce an error, the attacker should insert some statements that evaluate to true or false (via an tautology) in every possible input field.

2. **What can be observed by the server response?**

   The attacker should notice that no error message is generated but the response from the server is different when inserted in the search field.

   The response from the server is displayed as the list of items or the JSON object that can be viewed in the developer tools. Since no error message is displayed but the content changes the attacker knows it is a blind sql attack.

   Learning objective: Understand the difference between a error-based and blind SQLi attack and understand the syntax for inserting malicious SQL in this scenario.

[4]https://www.w3schools.com/sql/sql_wildcards.asp

1.1.3 Task: Password Hack

After the warm-up task, this task aims to implement an effective SQL injection to read from the database. The goal of this task is to access the admins password which is stored as a not-salted MD5 hash in the database.

1.1.3.1 Special Notes

- The task can also be solved with a binary search instead of the UNION operator. Therefore the password must be guessed character by character.
- When using the UNION operator: Each select statement within must have the same number of columns and each column must have similar data types. Further the columns must be in the same order.

1.1.3.2 Guide

The task can be guided as follows including additional hints to achieve the goal and understand the basics of the attack:

1. **How is the database organized?**

   In order to retrieve information from a database you must be aware of the underlying database schema i.e. the tables, attributes and relations. Basically in such attacks the required table and attribute names must be guessed by the attacker. However, it is often the case that the admin is stored as the first user in the database and therefore have attributes such as the username *admin* and id 1. First, try to insert a query to find users with the username admin and observe the number of attributes (number of columns) that are stored in one row.

   Learning objectives: Find entry point to the database for SQLi attacks.

2. **How to get the required information?**

   Basically the scenario can be solved with a binary search by collecting the password character by character or either with the UNION selector. The UNION keyword lets you execute one or more additional SELECT queries to the *original* SELECT query of the search string i.e. the original search query is in form of *SELECT inputstring FROM table*. This additional SELECT statement appends the results to the original query i.e. the query that takes the search string and performs a SELECT operation on the database. Two conditions must be met: both tables must have the same number of columns as well as the same datatypes. Therefore you have to observe the previously mentioned JSON object.

   Learning objectives: Understand how the query for the displayed items looks like and abuse of the UNION operator.

3. **What security mechanisms can be used to store passwords?**

   Passwords are usually not stored in plain text in the database for security reasons. Even if an attacker is able to hack the database and get the passwords from users, the attacker will not be able to use the hashed password for a login, etc. Traditional encryption algorithms include the md5 algorithm (which is not recommended because of security gaps.) The usage of md5 can be guessed by the byte length of 128 bytes. The hashed password can be decoded in an online decoder because it is unsalted i.e. no additional string is appended to increase the entropy of the password value.

Bibliography

[BK04] Stephen W Boyd and Angelos D Keromytis. "SQLrand: Preventing SQL injection attacks". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2004, pp. 292–302.

[H+06] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. "A classification of SQL-injection attacks and countermeasures". In: *Proceedings of the IEEE International Symposium on Secure Software Engineering*. Vol. 1. IEEE. 2006, pp. 13–15.

[HO05] William GJ Halfond and Alessandro Orso. "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks". In: *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM. 2005, pp. 174–183.

[Hot04] Cameron Hotchkies. "Blind SQL injection automation techniques". In: *Black Hat Briefings* (2004).