3 SQL injection

The following exercise will help you understand the SQL injection techniques and systematic vulnerability testing. While the main focus is on the SQL injection, there is also one Insecure Direct Object References vulnerability that you are supposed to exploit.
You will likely need an intercepting proxy for these assignments, read section Preparation for further details.

For this part of the lab exercise, a Hackxor webapp hacking game is being used. The game consist of a set of sites that you should hack. The first two steps of this hacking game, which include *wraithmail* (an e-mail service) and *cloaknet* (an anonymizing proxy service), are used for this exercise.
For those of you who might want to try their skills directly, we encourage you to try solving the first two steps (wraithmail and cloaknet) of the game without reading the lab instructions in the following section. However, you will still have to answer the questions anyways at the discretion of the lab assistant that examines your work.

The game scenario is as follows: you are a hacker who got the task to track down a person who performed an attack on one of the wraithmail accounts; for that, you should login into wraithmail.csc.kth.se:8080 and read the letter with the "trace job" description.
Your *login* credentials are:

User `algo`

Password `smurf`

3.1 WraithMail

This section contains the Insecure Direct Object References vulnerability. You are more than encouraged to try any other attack to check whether the site is vulnerable to any of them or not.

Important hints:

- Pay attention to the *Referer* line in the "trace job" letter
- Explore the site, look for a vulnerable add-on
  Note: You will need an intercepting proxy to exploit the vulnerability.
- If your attack succeeds, you will be able to see that the *abuse contact* is from the *cloaknet*

Answer the following questions:

- Briefly describe the attempt attack of the hacker that you are tracking (no details required, just the rough idea).

  _____

- What username was used for the attack?

  _____

- What was the logged IP address?

  _____

Milestone:
Report your progress to a lab assistant _____

3.2 Cloaknet

By now, you have got evidence that the attacker used a cloaknet proxy to hide its IP. Now, your task is to perform an SQL injection attack to retrieve the account credentials (login, password, id,...) of all registered users at cloaknet.
You should also be able to confirm that the source of the attack was from GGHB, that the target was *wraithmail:8080/send.jsp*, and that the date matches the one you could see in the wraithmail logs.

In order for you to find out who the attacker is, you should start with the Cloaknet website: cloaknet.csc.kth.se:8080. You might need to look at the SQL Injection Cheat Sheet mentioned in section Preparation to complete this task.

Guidance with questions:

- Name at least 3 ways (in terms of the HTTP protocol!) that are used by web applications to get input data and that might be susceptible to injection.

- A database has many different data types, but all of them can still be classified into a few groups based on how they are represented in the SQL statements. Name these groups.

- One of the values that goes as input to the SQL query in Cloaknet is *Username*. Name two other values that might be used as input in the SQL queries (**Hint:** these 3 values cover all "groups" from the previous question).

- How do you think the cloaknet's SQL queries might look like?

- Check whether the cloaknet web application's inputs are vulnerable to SQL injection or not. How do you do that? Give two examples.

- Were you able to cause an SQL error or an HTTP error (500) to be displayed on the webpage? How did you do it? What information did you learn? What kind of database and web server are being used?

- Why is it useful to know which database is used?

- All inputs are filtered (the filter is the same for all)! The filter escapes some characters and removes some words (e. g., *SELECT*). Moreover, one of the inputs in cloaknet is treated with many characters as delimiters (even spaces). Why? Name at least four techniques to bypass filters in general (with examples) (**Hint:** check the cheat-sheet and try using predictable names for the attributes of each entity, that is the column names for

each table you want to use).

- The *UNION* operator is used to combine output from several *SELECT* statements. What requirements should these statements meet in order to be "unioned"? What if the table you want to union does not have as many columns as the first? How do you solve this?

- You should be able to infer the number of columns used in the query statement from the output you see on the page. Sometimes, though, not everything is shown, and then it is important to find out the number of columns. How is it done? Provide an example.

- You can *UNION* tables even without listings column names. How can you include all columns of a table in the *UNION* without explicitly naming them?

- What are the username and the password of the attacker?

Milestone:
Report your progress to a lab assistant _____

3.3 Protection against SQL injection

There are many mitigation techniques for SQL injection.

Questions:

- Describe how parametrized queries help protect against SQL injection. What other protection techniques can be used on the application's code level?

- What is a stored procedure? Describe how it protects against SQL injection. What other protection techniques can be used at the database level?

- What combination of protection techniques would you choose, to mitigate the SQL injection threat?

Milestone:
Report your progress to a lab assistant _____