Security I: Common software vulnerabilities: SQL injection

Required submission: Enter all three recovered flags at the flag checking portal.

Introduction

Before tackling the following stages, you will need to be familiar with Structured Query Language (SQL), particularly SELECT statements. You will also need to understand how SQL queries can be subverted when user-supplied input is not robustly sanitised. All three stages require access to a simple web application which exhibits a number of SQL-injection vulnerabilities. On completing individual stages, you will retrieve a flag (a string of 8 alpha-numeric characters). The flags must be submitted to the flag checking portal. If you submit a correct flag, your achievement will be logged automatically.

Rules of engagement

These instructions request that you attack a web application, and it is important that you only attack the designated targets. All target web pages will have the title "Valid target", so please check for this first. Please do NOT attack the University's Raven service.

1. Authentication bypass

Begin by accessing the Raven-protected page at: Stage 1. The page presents a simple login form. You must bypass the login form using SQL injection, and login as user "admin". The SQL query which is being executed by the web application when you click "Login" is presented at the bottom of the page to assist you in developing a functioning exploit. No flags are available in this stage, but you will need to be able to login to the application to gain access to subsequent stages.

2. Modify search query

After logging in to the web application in Stage 1, you will have been redirected to a simple search page. The search page allows you to find dinosaurs by name. Some dinosaurs are classified, and their details are not normally included in

the search output. You will need to modify the executing SQL statement appropriately in order to return additional classified results. Once this has been accomplished, the first flag will be revealed (in blinking text).

3. Query adjacent tables

A flag has been stored outside of the table "dinosauria" on which the search page bases its queries. The flag has been assigned to a user variable, "@flag". You will need to modify the executing SQL statement appropriately in order to include "@flag" in the results. Once this has been accomplished, the second flag will be revealed (in blinking text). This example query should help you with the correct syntax:

```
SELECT @flag;
```

If you are able to retrieve the flag, then you should also be able to find out what the admin user's password is for the web application!

4. Blind SQL injection

A second page in the web application displays some dinosaur statistics. It queries the same table "dinosauria", but rather than listing all the rows returned, it counts how many dinosaurs lived in a particular age and displays only a single number. The page is vulnerable to blind SQL injection. A flag has been assigned to a user variable again, "@flag". You will need to modify the executing SQL statement appropriately so that you can submit yes/no queries to the database management system, and this should allow you to exfiltrate the flag. These example queries illustrate how you might submit a yes/no query to the database management system:

```
SELECT genus FROM dinosauria WHERE genus="Tyrannosaurus";
+---------------+
| genus         |
+---------------+
| Tyrannosaurus |
+---------------+
1 row in set (0.00 sec)
```

One result was returned as expected.

```
SELECT genus FROM dinosauria WHERE genus="Tyrannosaurus" AND 1=0;
Empty set (0.00 sec)
```

No results were returned, because the RHS of the test was false.

```
SELECT genus FROM dinosauria WHERE genus="Tyrannosaurus" AND 1=1;
+---------------+
```

```
| genus         |
+---------------+
| Tyrannosaurus |
+---------------+
1 row in set (0.00 sec)
```

One result was returned because the RHS of the test was true.