# Sniffing Lab Instructions

## Part 1: Sniffing

This part of the lab involves intercepting login credentials to a server. This server will be run locally on your Kali VM, and you will be using Wireshark to intercept the login credentials. The server is using the insecure HTTP protocol to transmit login credentials and as a result, login credentials are sent in the clear. This is possible since HTTP has no confidentiality, and why you should be using HTTP with TLS v1.3 (HTTPS).   
  
Sniffing Lab Part 1 - Lab Setup:

1. Use Kali Linux

>$ sudo apt install git pip

>$ git clone https://github.com/ancilchandler/HVH\_Sniffing.git

>$ pip install flask

>$ sudo apt install siege

Sniffing Lab Part 2 – Setting up Infrastructure:

1. Navigate to the recently cloned Github directory

>$ cd HvH\_Sniffing/

2. Run flask server. This is the server that is running the HTTP protocol.

>$ python3 serv.py

2a. If you can't execute it, do >$ chmod +x serv.py

This is the server running the application. Our goal is to intercept the credentials being sent to this server. Minimize this terminal.

3. Note the IP address and port number of your flask server. The picture below is an example of what you should see. Since you are running this locally, either IP address will work.

A screen shot of a computer program

Description automatically generated

Sniffing Lab Part 3 – Setting up “User”:

Set up the "user"

This is a simple Python script that mimics a user logging in. It sends an HTTP request and attempts to login to the admin panel every 3 seconds on your flask server.

**NOTE #1: This code requires a url as a command line argument. Reference the example below if you are confused as to what this is asking. It needs a ip address followed by a colon and then a port number. By default, it is port 5000.**

**NOTE #2: If you are having trouble and an error message pops up as something like “port 5000 already in use”, you will need to kill your user process. The following commands will help with that:**

**Identify the PID of your process using port 5000**  
>$ sudo netstat -tuln | grep 5000

OR

**>$ sudo ss -tuln | grep 8080**

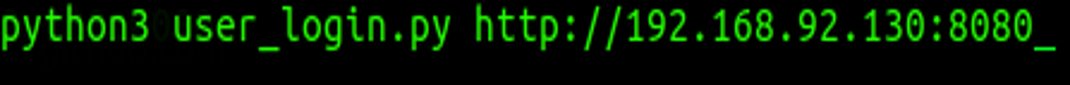
**THEN**

**sudo kill -9 <INSERT\_PID>**

Your goal is to intercept this login with Wireshark and login as the user. For <insert\_ip\_address> and <insert port number>, use the ones from your flask server.

>$ python3 user\_login.py http://<insert\_ip\_address>:<insert port number>

Example:



After you have completed this, the user script should now be running, and your “user” will be attempting to login into the flask server with admin credentials.

Sniffing – Lab Setup part 3

1. Open Wireshark, scan for all connections (any)

**>$ sudo wireshark**

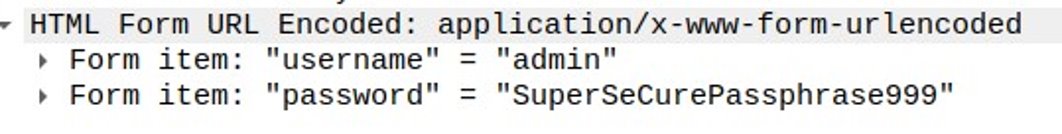
2. Filter by HTTP protocol. Filter: http. Reference example below.

2a. (OPTIONAL) Filter out dns and icmp. They can make your screen cluttered. Reference example   
 below.

A screenshot of a computer

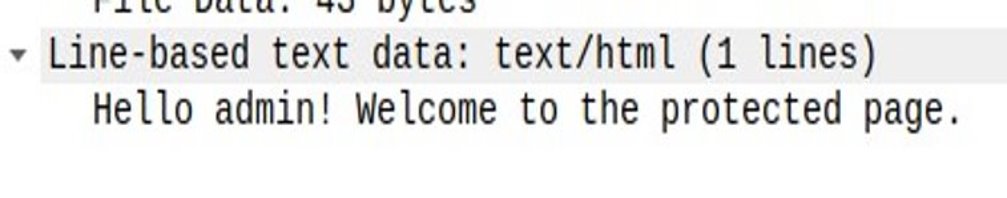
Description automatically generated

3. Look for admin credentials. Reference example below:



4. Login to server ---> screenshot #1. Reference below for an example.

Wireshark Example:



OR

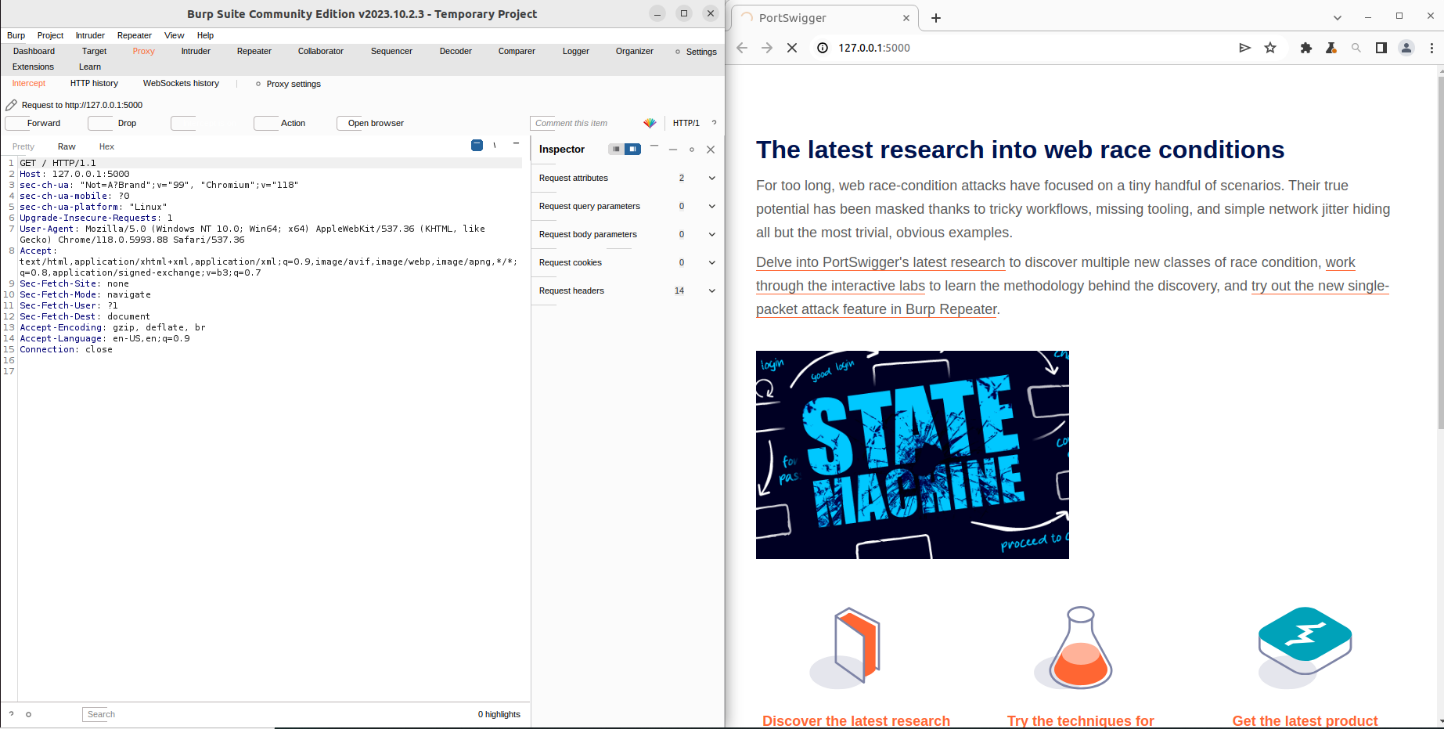
Browser Example:



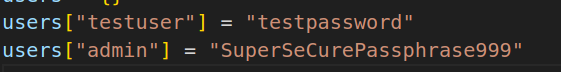
## Part 2: Session Hijacking

This part of the lab involves using the flask app that’s part of the zip as well as burpsuite. Open up the browser inside of burpsuite under the proxy tab. Run the flask app and navigate to the ip it gives you in the address bar. You will have to forward all of the requests the client makes. Use one of the accounts that is given in the top of the flask app. Login as admin or test\_user. Observe what changes in the data that burpsuite sends. There is a vulnerability in the cookie issuing logic. Your screenshot #2 is proof that you logged in as target\_user

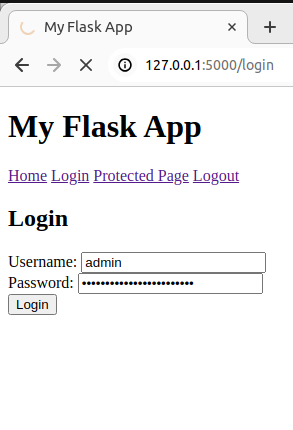
Walkthrough:

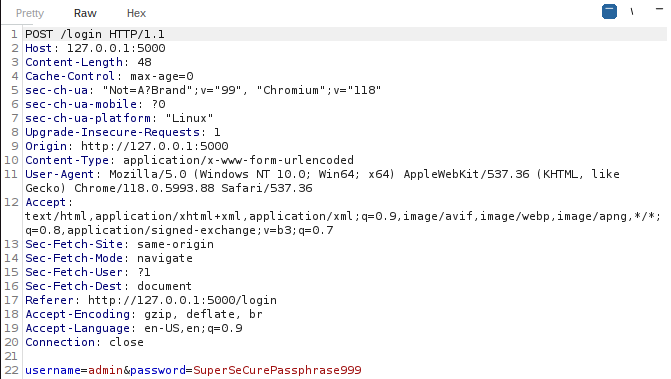


Open the browser on the right side of your screen and burp on the left.



Navigate to /login and login as one of these two users. You might need to turn on interception on the left and press forward on each request.





This is what the POST request looks like for the login page. Go ahead and forward this.



Notice that the server now issues you a cookie. If you select the string in red, it will give you some interesting info. Copy the whole line of Cookie: user\_id=””. You’ll need it later. Now navigate through to the logout. Forward as needed.

If you look at the top of the flask app source you can see three users. Target\_user has a long hashed password as opposed to the plaintext the other two have. The screenshot for this part is the /protected page with authentication as target\_user.

Further hint: The cookie is the username of the user with base64 encoding. If you logout then try to navigate to the /protected page, but replace the base64 string of admin or test\_user with the base64 of target\_user, you will authenticate without using the password for this user. I recommend using cyberchef to convert ascii to base64 and vice versa.

<https://gchq.github.io/CyberChef/>

## Part 3: Denial of Service (DOS)

The objective of this part is to perform a denial-of-service attack against this server. You will be using a suite of server stress testing tools called “siege” along with a piece of software from siege called “bombardment”. Siege is primarily used to benchmark and stress test servers and is an easy-to-use DOS tool against HTTP servers. Bombardment runs multiple instances of siege but with an increasing number of users overtime. While both are used for legitimate purposes, they have been repurposed to perform DOS attacks.

Your goal is simple: perform a DOS attack against this server, overloading it by sending as many HTTP requests as you can. The benefits of this attack are twofold:

1. Resource Starvation: You are overloading the server by sending a bunch of HTTP requests, thereby capping resources on your server hardware
2. Denial-of-Service: By overloading (or even crashing) the server, you are depriving legitimate users from logging in, thereby creating a DOS for this application

**NOTE: This part MAY OR MAY NOT** work, as there were mixed results. Your third submission is to write a small paragraph explaining why the results worked/did not work for you and why.

Denial of Service part 1 – Preparing the attack:

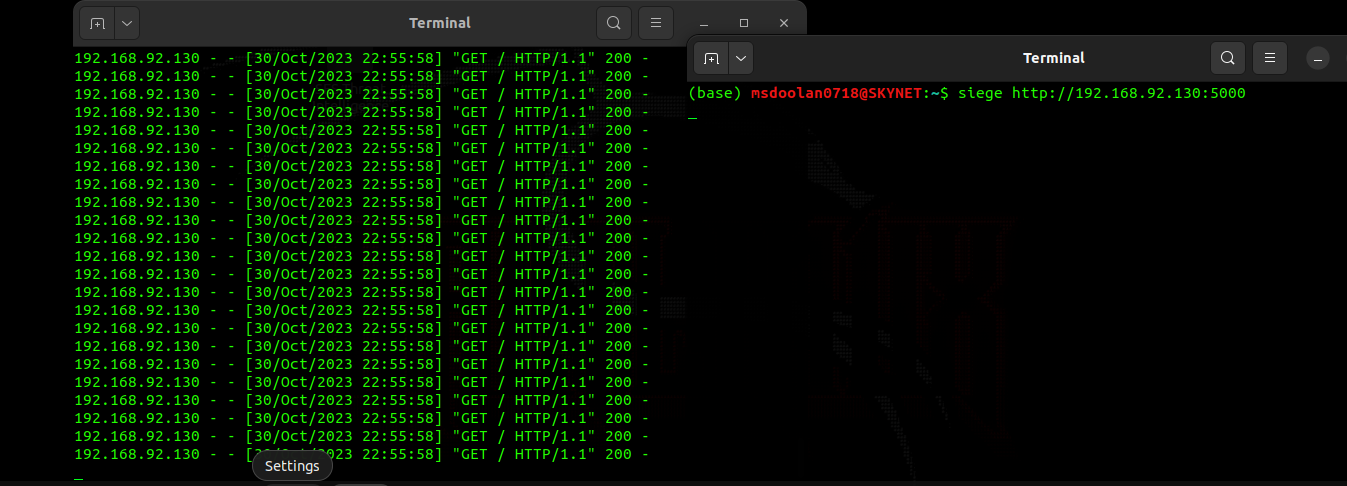
1. Ensure that your server is still up. Repeat the first part of this lab if you closed your server
2. Ensure that siege is installed

>$ sudo apt install siege

Denial of Service part 2 – Siege the Server:

1. Start with a basic test of your servers capabilities. Ideally, you want to see if this attack is even working. The following command is a basic load test against the server. Look at your server terminal to see if the attack is working.

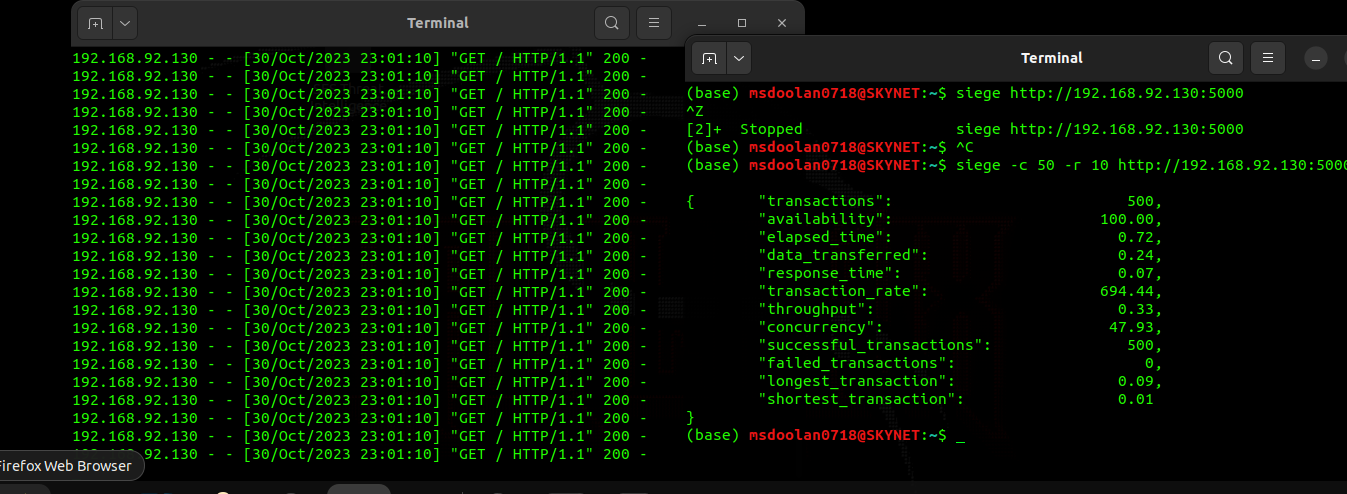
>$ siege http://<your\_server\_url\_here> :<insert\_port\_number>



As you can see, siege is very much working. However, the server is still alive. Also, you may not notice your application slowing down when you navigate the website. This is because only one user is trying to send HTTP requests. Let’s simulate even more users.

2. This is a more aggressive test. The –c switch specifies how many users and the –r specifies the amount of repetition each user will attack the server at. For example, 50 users x 10 repetitions means 500 http requests per attack. Look at your HTTP output and note the results.

>$ siege -c 50 -r 10 -u http://your\_server\_url\_here



As you can see, the server is still up. Clearly, this is not working. Let’s use another tool called bombardment, which is a tool that comes with siege. This is a far more aggressive version of the previous command. Bombardment will start with the specified number of users but will slowly increase the number of users and repetitions until the server (or your computer) crashes. This is used for extreme stress testing on servers and also to simulate DOS attacks.

**NOTE: (OPTIONAL) Bombardment only allows 255 users without root privileges. Anything above this needs root AND the limit needs to be manually changed in the config file. To change this, follow these commands:**  
  
**Locate the Configuration File: The default configuration file for siege is usually located at ~/.siegerc for your user or /etc/siege/siege.conf for system-wide settings.**

**Edit the Configuration File: Open the configuration file in a text editor. For example, using nano:**

**>$ nano ~/.siegerc**

**OR**

**>$ sudo nano /etc/siege/siege.conf**

**Modify the Configuration: Find the line that starts with limit = (it might be set to limit = 255 by default). Change the value to the desired number. For example, to increase the limit to 500:**

**limit = 500**

**Save and Close: Save the changes and close the editor. If you're using nano, press CTRL + O to save, then CTRL + X to exit.**

Denial of Service part 3 – Bombard the Server:

3. Run bombardment. Bombardment requires a target in the form of a text file. Run the following   
 command. This command appends your server url to a text file for bombardment to use.

>$ echo "http://<insert\_server\_ip>:<insert\_port\_number>" > urls.txt

Then run bombardment with the following commands. I would recommend lowering the values if this takes too long. Reference below for an explanation of each number.

>$ bombardment urls.txt 10 10 1000 1

Explanation of each number:

1. The first category is the number of initial clients attempting to attack the server (10 users)
2. The second category is the incremental value of users after each failed attack (+10 per failed attack)
3. The third category is the max number of attacks allowed for one command (1000 max attempts to attack before siege terminates)
4. The fourth category is the delay (measured in seconds) of each attack (1 second delay)

4. Submission #3 ----> Analyze results. While launching the attack, try to login and move around your server. Look at your terminal and see if your server crashed. Provide a short paragraph explaining why you THINK the attack worked/did not work in bringing down your flask server. My server stayed up even after that.

