Difficulty: Assorted, indicated by different steps.

Learning Outcome: Students will learn about how HTTP requests are crafted web browsers and use authentication mechanisms and/or authorization tokens to access protected web pages.

GitHub Classroom:

Description:

Alice needs to authenticate to the web service using port 380 on the host hw2.csec380.fun. In this assignment, you will write several python scripts that will allow Alice to authenticate to a remote web server using only sockets and strings without many libraries that were usable in previous assignments. Alice's username will always be *alice* and Alice's password will always be *SecretPassword123!*. Each step of the assignment will get progressively more challenging. You must submit a different Python script for each step. Each python script must print the last HTTP response your script received in its entirety.

With the exception of Step 1 and Step 2, each step is complete when you receive a 200 HTTP response containing the message "*Welcome Alice!*". Pay close attention to the HTTP responses you receive. The web server has been build to provide error messages – which can sometimes be a little silly – that inform you of common mistakes. All attempts to log in must be automated.

Note – hw2.csec380.fun can only be reached if you on the RIT campus (your traffic must originate from a 129.21.0.0/16 IP) or from one of the RIT VPNs (all of which have public IP addresses in the range 10.0.0.0/8). It is assumed that you will come to campus or connect to the campus VPN in order to complete this homework service. If you need another IP address allowed, please reach out to me with the IP address needed to be allowed and the reason.

Step 1 – Hello World
Difficulty – Tier-0
Points: 10
Endpoint: /hello

Create a simple Python script that accesses the /hello endpoint on the application. This endpoint will always return an HTTP response containing the text *Hi!.* A sample Python function that accomplishes this has been provided as a starting point for the assignment.

```python
CRLF="\r\n"

def calibrate():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((target_host, target_port))
    firstline = "GET /hello HTTP/1.1" + CRLF
    request = firstline + CRLF
    client.send(request.encode())
    httpresponse = client.recv(8192)
    response = httpresponse.decode()
    httpresponse = client.recv(8192)
    response = response + httpresponse.decode()
    client.close()
    print(response)
```

Note that this sample code has one socket send and two socket reads, which are concatenated to form a complete HTTP response. As of 9/1/22, the web application (based in Flask) is including a null byte in the middle of a correctly structured HTTP response for unknown reasons. "In the real world", you should only need one client.recv() to fetch a full HTTP response. This holds true for all endpoints used in this assignment.

The goal for this is step is to implement this Python script and validate that you are able to receive a correct response.

Step 2 – [Not so] Secret Agent
Difficulty – Tier-0
Points: 10
Endpoint: /test

Unfortunately, the site that Alice must authenticate to only permits a custom user-agent header. The value that user-agent header must contain is "CSEC-380". Fortunately, there has been enough difficulty with implementing this that the site provides an endpoint to validate your user-agent header. If you set your user-agent correctly, you will receive an HTTP 200 response code containing "Working as Expected".

Step 3 – BASIC Training
Difficulty – Tier-1
Points: 10
Endpoint: /basic

Alice has informed us that they must use the basic authorization header to access the endpoint */basic*. Research the structure of the basic authorization header and use the provided credentials to generate the correct authorization token. Your Python script does not need to generate the

authorization token; calculated offline using other tools and hardcoded. The authorization header will only be needed for the Python script created for step 3.

Step 4 – GET with it
Difficulty – Tier-2
Points: 20
Endpoint: /getLogin, /getSecurePage

Alice has informed us that the web development team has seen the light about the insecurities of basic authentication and has decided to "improve" their security. Rather than using the authorization header, the credentials are supposed to be passed by two parameters: username and password. Once those credentials are presented to the */getLogin* endpoint, a session will be established and the user will then be able to access */getSecurePage*.

Step 5 – Mission Impossible: POST Protocol
Difficulty – Tier-3
Points: 20
Endpoint: /postLogin, /postSecurePage

Including user credentials in GET requests is not ideal? Alice has let us know that the company was shocked to discover this. They've also let us know that the company has changed their authentication workflow to require that parameters be passed via POST instead of via GET. Otherwise, the parameters – *username* and *password* – are the same. Be careful that your get your Content-Length header correct, however!

Step 6 – Waiting for Godot
Difficulty – Tier-3 [same as step 5]
Points: 5
Endpoint: /delayedPostLogin, /delayedPostSecurePage

Alice has let us known that they received an email from the company complaining about people trying to "attack their site" with "automated logins". To address this, the company has added some code to slow down the attackers. That's fine! With web, as in all things, it can be better to take your time anyway. Modify your script from the previous step to bypass this control. The only difference between step 5 and step 6 is timing.

Step 7 – The Keymaster
Difficulty – Tier-3
Points: 10
Endpoint: /jsonLogin, /jsonSecurePage

Alice, having signed up for the company's development program, has found out the company is trying to get with the times and set up a new-fangled API to allow the users that want to automate their access to do so without using other endpoints. If you make a POST request to the /jsonLogin endpoint with the same parameters as the last two steps – username and password – you will be issued an API token contained inside a JSON response. That API token can then be provided as the value for the POST variable *apikey* to access the /jsonSecurePage endpoint without needing to provide other usernames or passwords.

"In the real world", you would be able to use the API key without providing any other pieces of information. However, due to implementation limitations on the assignment, you must include the session cookie set by /jsonLogin as you have in steps 4-6. This is used as an anti-cheating to ensure that api tokens are not reused. Note that the API key should only be used once.

Step 8 – Evading 'captcha
Difficulty – Tier-4
Points: 15
Endpoint: /captchaLogin, /captchaValidate, /captchaSecurePage

Alice has let us know, sadly, that they received an information that the company is – once again – discontinuing ability to automate logins. To address this issue, the company is adding a "Captcha" – a problem the user must solve in addition to providing real credentials – in order to access the secure page. The user provides usernames and credentials to /captchaLogin via POST requests and will receive a 200 response including a mathematical problem that must be solved. There will always be exactly two numbers between 0 and 100 in the response and exactly one mathematical operator. Solving the captcha can be accomplished evaluating the result of the operator applied to the two operands. There may, however, be garbage in the response intended to make it difficult for non-humans solve.

The result of performing the mathematical operation can be provided to /captchaValidate in the POST parameter *solution*. Once /captchaValidate has checked the provided solution against the correct answer and confirmed it is correct, you will be able to issue a request for /captchaSecurePage.

Step 9 – Bugging Out
Difficulty – Tier-5
Points: 10 [Extra Credit]
Endpoint: Any

Alice has let us know that the site has announced the creation of a bug bounty program! Anyone who submits a bug bounty documenting unintended security issues with any of the above authentication workflows – particularly issues that allow credentials from one workflow (such as GET authentication) to be used on another workflow (such as POST authentication) – will receive a prize! Each report that is reproducible can earn the submitter 10 extra rewards points [ie: extra credit]. Note that sufficient information must be provided to allow the reviewer to replicate the bug. Denial of service bugs are considered out of scope and should not be attempted.

Requirements:

- You may use the socket library, the time library, and the json library. Other libraries must receive special permission.
- You may use libraries that allow for regular expressions to be performed on strings but you may only use their regular expression capabilities.
- You may not use libraries that generate HTTP requests for you (such as requests or urllib)
- You may not use libraries that parse complete HTTP responses for you (you may use JSON parsing libraries for parsing the bodies of some HTTP responses)

Deliverables:

a. By 8:00 PM on 09/15/2022, you must submit the following to the DropBox on MyCourses:
   a. A copy of your Python scripts
      i. A zip is fine if MyCourses gives you issues uploading .py files.
      ii. This will be used to ensure your GitHub repository has not been changed.
   b. A PDF containing screenshots of each of your scripts printing the final HTTP response in the step.
   c. A GitHub classroom link to the repository containing your submission. The contents of this GitHub repository must match your uploaded file.