Exercise 11

Web vulnerabilities and "Not so small" applications / `ex-web`

Log into your VM (`user` / `1234`), open a terminal and type in `infosec pull ex-web`.

- When prompted, enter your username and password
- Once the command completes, your exercise should be ready at `/home/user/ex-web/`

When you finish solving the assignment, submit your exercise with `infosec push ex-web`.

Background and Setup

After surviving the horrors of the shellcodes, the packet drops of the network, and the logical vulnerabilities of Bob, you feel ready to look for a real job. Within a few days, you manage to get your first role, with the details as follows:

- Your name is Edward Hailden (hi Edward!)
- This is your first day at your new work - the NSK
- You are a **highly-skilled** and **overly-motivated** worker
- The organization recently ditched its old email-system, and move to the new-shiny chat platform - **"Slack-off"**

Now that you know your role, let's setup the exercise:

- Enter the exercise directory
- Run `python3 -m server` to launch the Slack-off system
  - Make sure it connects and no error message is printed
  - **Make sure you are connected to the internet**, otherwise some functionality may fail
- Open the browser and navigate to `http://localhost:8000`
  - Make sure you see a login screen

IMPORTANT: If you need to reset the exercise (clear the login and messages), you can navigate your browser to `http://localhost:8000/reset`

IMPORTANT 2: The exercise will not work if you use the browser in "private browsing (aka incognito). Use the browser regularly, and "clear all data" using the reset link if needed.

Special instructions

- In this exercise, you don't need to write any Python code, you only need to find and exploit vulnerabilities
- Since we need to know how to use the exploits, **please pay attention!**
- There are two ways to exploit vulnerabilities in the system - the first is by sending a maliciously crafted message on one of the chat channels, and the second is by running custom code in the JS console
    - Use custom messages when the vulnerability is related to how messages are handled
    - Use the JS console to alter the way your browser interacts with the server (for example - modify existing forms, change cookie values, etc.), or to trigger and exploit other vulnerabilities (for example, by crafting custom web requests)
- In each question in this exercise, you need to identify and exploit a vulnerability in the system, to perform a specific action. To exploit the vulnerability, you will use **exactly one** of the two suggested ways:
    - **Option 1**: If your solution is based on crafting a malicious chat message, then it should work by sending a **single message**. In this case, create a file named `qX.msg` where:
        - Replace `X` with the question number
        - On the first line of the file, put the channel to which the message should be posted (the channel should begin with #)
        - The rest of the lines are the content of the message
    - **Option 2**: If your solution is based on using the JS console, create a file named `qX.console` where:
        - Again, replace `X` with the question number
        - The file contents is the code we need to run in the JS console
- As usual, document your solution in `qX.txt`
    - Your description should be short and concise!
    - 2-15 lines should be more than enough
    - Too long explanations that don't get to the point, will lose points
    - Solutions without an explanation will not be graded (i.e. get 0)

For example - a solution for question numbered 123 of "how do you join a channel?" would be:

- `q123.msg` - Will contain

```
#general-spam
/join #foo
```

    - We put a random (existing!) channel name, because it doesn't matter on which channels this message is sent
    - Even if the exact channel doesn't matter, don't skip it - pick some existing channel
- `q123.txt` - Will contain `According to the manual, this is the command to join the channel #foo`

Finally, before starting this exercise, re-read the recitation, especially the last part (learn to fly). Read it, make sure you understand the concepts, and then read it again. **Don't waste time reading code that you shouldn't care about(!).**

Question 1 (20 pt)

Edward is super-excited about his new job at the NSK. So excited that he got there at 07:00 am on his first day, before the IT was there to reset his password. But Edward knows better than to slack-off™ for 3 hours - he will login to the system anyway!

1. Find a vulnerability in the login process that will allow you to log-in the user `edward` without a password
2. In `q1.console`, write the JS console command to perform the login
    a. We will refresh the page after executing this command, and then we should be logged-in
3. In `q1.txt`, document the vulnerability and explain how/why your JS command works

Notes:

- Before moving onto the next questions, **read the tutorial in the #system channel**, and try experimenting with the Slack-Off system!
- If you don't manage to solve this question, note that it's possible to solve the rest of the questions with a different user.
    - We will not tell you which users and how, the details are in the code :)

Question 2 (30 pt)

Edward wants to know when his boss arrived. However, he knows nothing about his boss except for his name (Bruce Summersteen) and his username (`boss`, of course).

1. Find a way to make every user that logs-in, send a message of Good Morning! to the `#general-spam` channel
    a. If a user is already logged-in, the message should be sent immediately after we perform our exploit

   b. It is OK if the message is sent again every time the user refreshes the page (or performs a log-in again)
   c. Your attack should work regardless of the channel the user is viewing!
   d. Hint: You can inject a script tag to solve this question
2. In `q2.console` / `q2.msg` write the console command to execute / message to send, and in `q2.txt` document the vulnerability and explain why/how your attack works

Question 3 (25 pt)

After lunch, Edward decides it's a good time to share a cat GIF with his colleagues. However, he wants the image to disappear once they view it - to make sure they don't waste their precious time.

1. Find a way to post an image to the **#general-spam** channel, so that it disappears (for that user) when the page is refreshed or when he switches channels!
2. It is OK if the message doesn't disappear, but the image should become invisible when the messages in the channel are reloaded
   a. To save your state, use `window.localStorage` instead of messing around with cookies
   b. In `q3.console` / `q3.msg` write the console command to execute / message to send, to send one cat GIF (chose a URL of any cat GIF from the internet) and make it disappear when viewed after the first time
      i. You don't need to support multiple images, or sending multiple copies of this image
3. In `q3.txt` document the vulnerability and explain why/how your attack works

Notes:

- Every submission that doesn't pass our automatic tests will be tested manually. Therefore, please make sure your GIF is funny :)
- Your solution should post a message (i.e. a new message record should be created in the DB).
  - If your solution is an XSS that shows an image, but no message was added to the DB, it is not a valid solution.
- **You may not use the same vulnerability from question 2**
  - If you attack the same piece of code (i.e. the same vulnerability), even if your attack (exploit) differs, you will not get any points.
  - In case of doubt, ask the TA by email (don't post solutions on the forum).

Question 4 (20 pt)

Bob forgot his password and is wasting precious corporate time while waiting for the IT. But, fear not - Edward to the rescue! Edward will reset Bob's password!

1. Find a way to reset the password of any user
2. In `q4.console` / `q4.msg` write the console command to execute / message to send, to reset the password of the user `bob` to `1234`. In `q4.txt` document the vulnerability and explain why/how your attack works

Question 5 (15 pt)

Edward wants to warn the IT of their lacking security, and he decides to do that by showing them that even if they'll remove JS from chat messages, it is still not enough. To do that, he decided he'll show them the risks of CSRF.

However, since Edward has high security awareness he updated his browser and so trivial CSRF attacks are harder to make (due to SameSite limitations), and so he can't give a demo to IT on the problems for employees with old browsers.

To allow demonstrating CSRF attacks, let's "downgrade our browser" by changing the cookie behaviour:

1. Open the file `firefox-howto.txt` included with your exercise
2. Follow the instructions in the file
3. Use firefox for the rest of this question (you'll need to log-in with some user to the chat system)

Now that we have an "old" browser:

1. Write an HTML file `q5.html` that when opened by anyone, will post on their behalf, to the channel `#announcements` the message Give Edward a Raise!
    a. You may assume those other users are logged-in
    b. You may use the template provided in the directory (you don't have to though)
    c. You will definitely encounter errors in the JS console (**think why!**), but this does not mean the attack won't work. The only way to check if the attack worked, is to check whether a message was posted
2. In `q5.txt` document the vulnerability and explain why/how your attack works

Final notes:

- Yes, the points sum to 110, it's intentional
- Don't waste your time, re-read the recitation
- Document the reasoning and rationale for your code (in addition to documenting the code)