XSS - Cross Site Scripting

Overview

Cross-site scripting (XSS) is a web security vulnerability that attackers can exploit in order to interfere with how a user interacts with a vulnerable website.

There are three types of XSS attacks:

- Reflected XSS - attack is mitigated via a HTTP request
- Stored XSS - attack is mitigated via the website's backend database
- DOM-based XSS - attack is mitigated via the client-side Javascript code

In this project we will be carrying out a reflected XSS attack and stored XSS attack.

Reflected XSS

Reflected XSS occurs when a website receives data in a HTTP request and 'reflects back' the data to the user. The website will not process or store this data, thus allowing an attacker to inject malicious content into the website.

One way an attacker can elicit a reflected XSS attack is to construct a malicious URL and, by way of a phishing email, trick the user into clicking on the malicious URL. This will direct the user to the vulnerable web page injected with the attacker's XSS payload which will execute in the user's browser. The attacker may try to mimic the login page of the website, and when the user attempts to log in via the fake login page the attacker can steal the user's login credentials.

Stored XSS

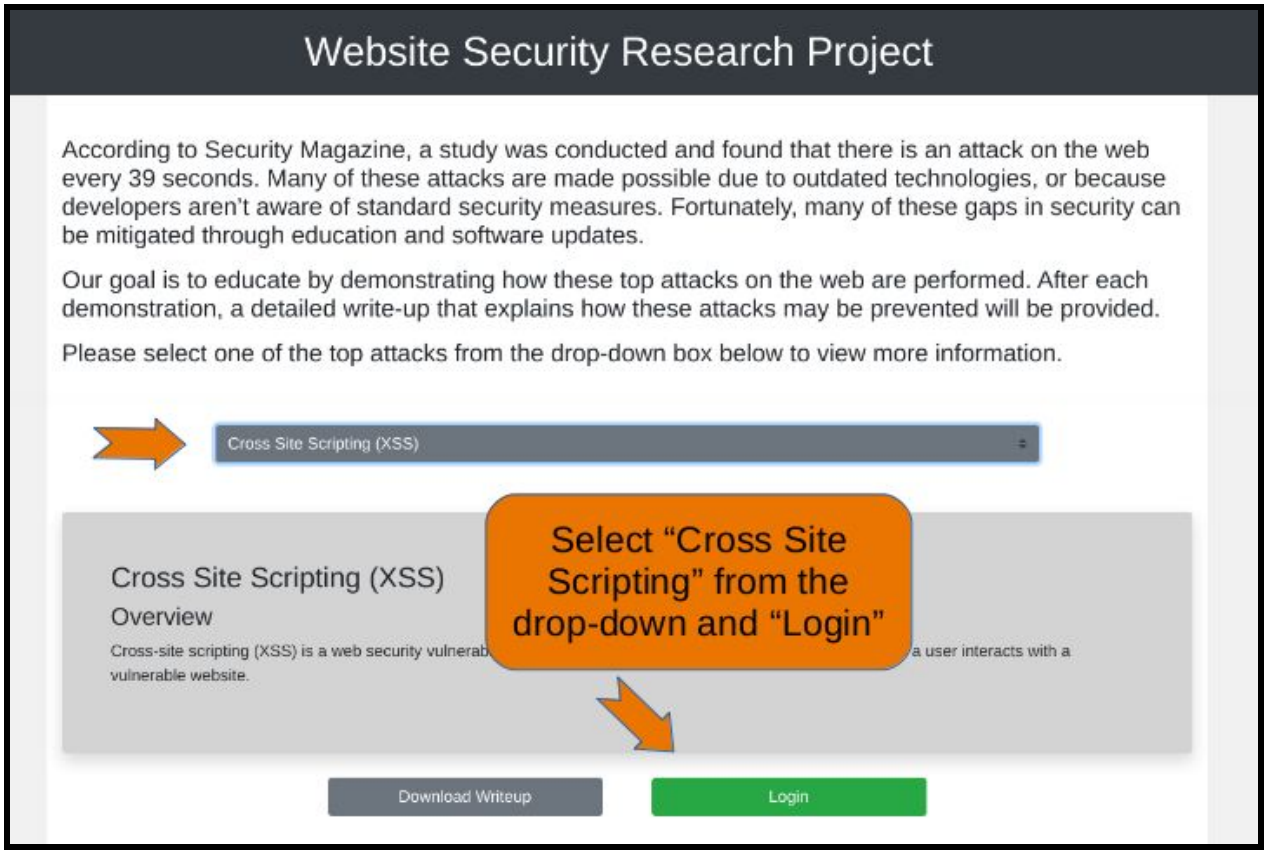
Stored XSS occurs when an attacker stores a XSS script on a server for later retrieval. The storage of the script can be in the form of forum posts, comments, usernames, or any other form of input that can be viewed at a later time. After the script has been stored, the attacker can carry out an attack by requesting the stored script from the server.

Dom-based XSS

Dom-based XSS occurs when there is client-side javascript that processes untrusted data from a user and writes it back out to the DOM. When a user is able to control what is displayed in the DOM, there is potential for scripts to be constructed by users that may cause harm. A good rule is to never use client-side javascript to populate the DOM with user supplied data.

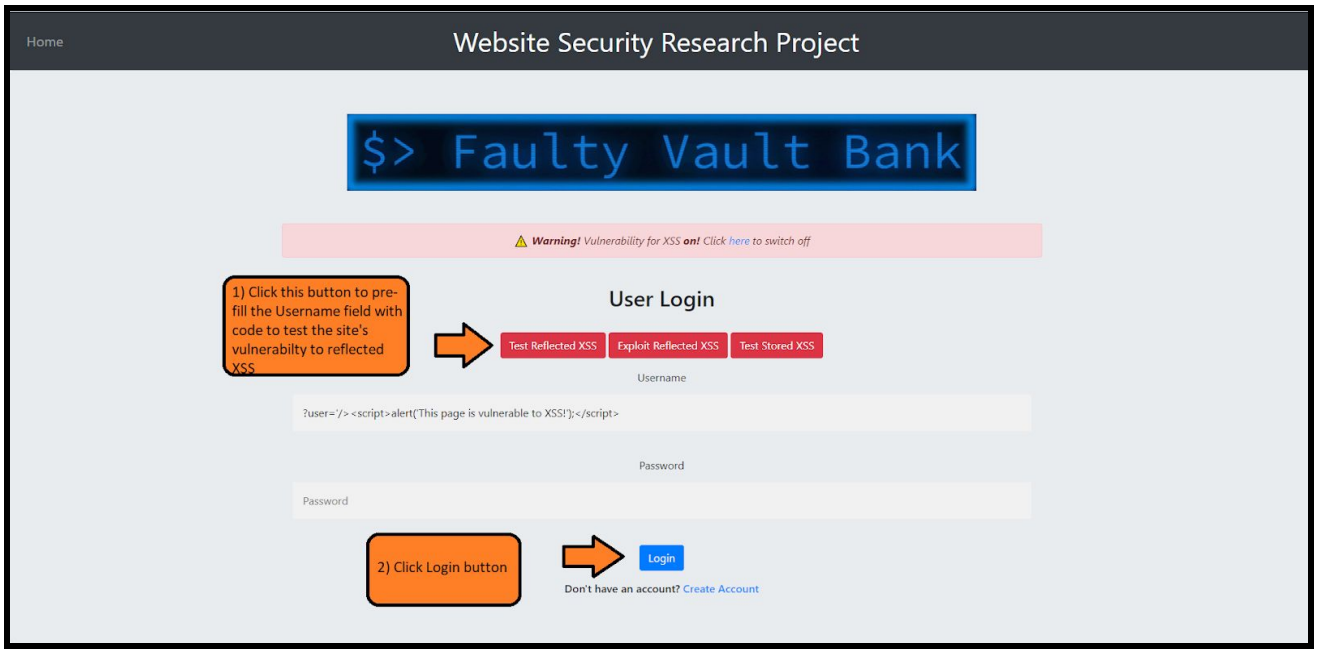
Attack Procedures

The screenshot below demonstrates the first step for all the XSS attack examples in this write-up. All remaining steps for each XSS attack can be found in their labeled sections.

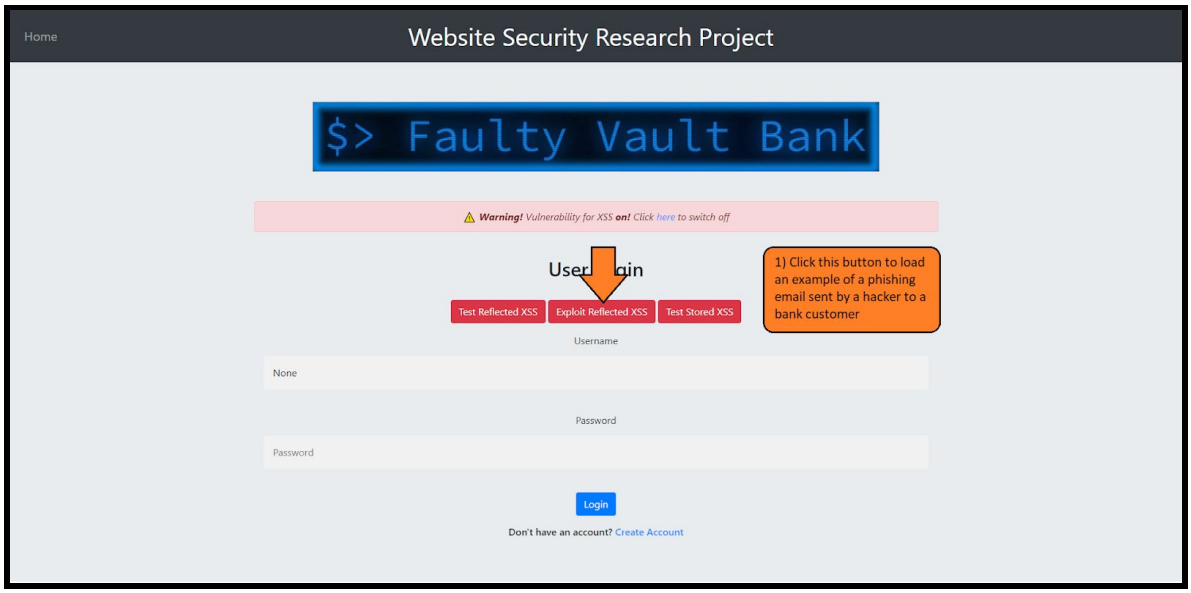


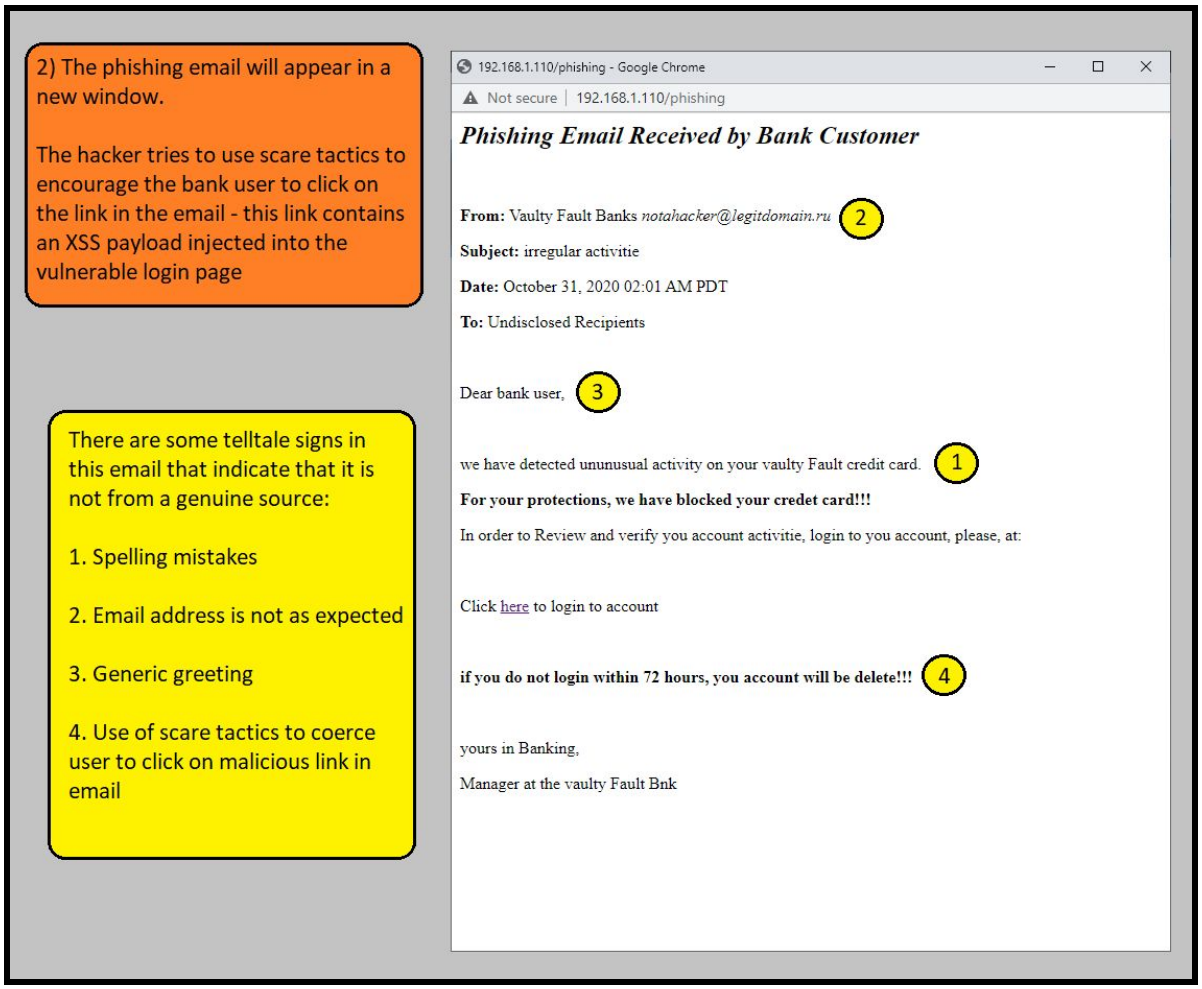
Reflected XSS (2 examples)

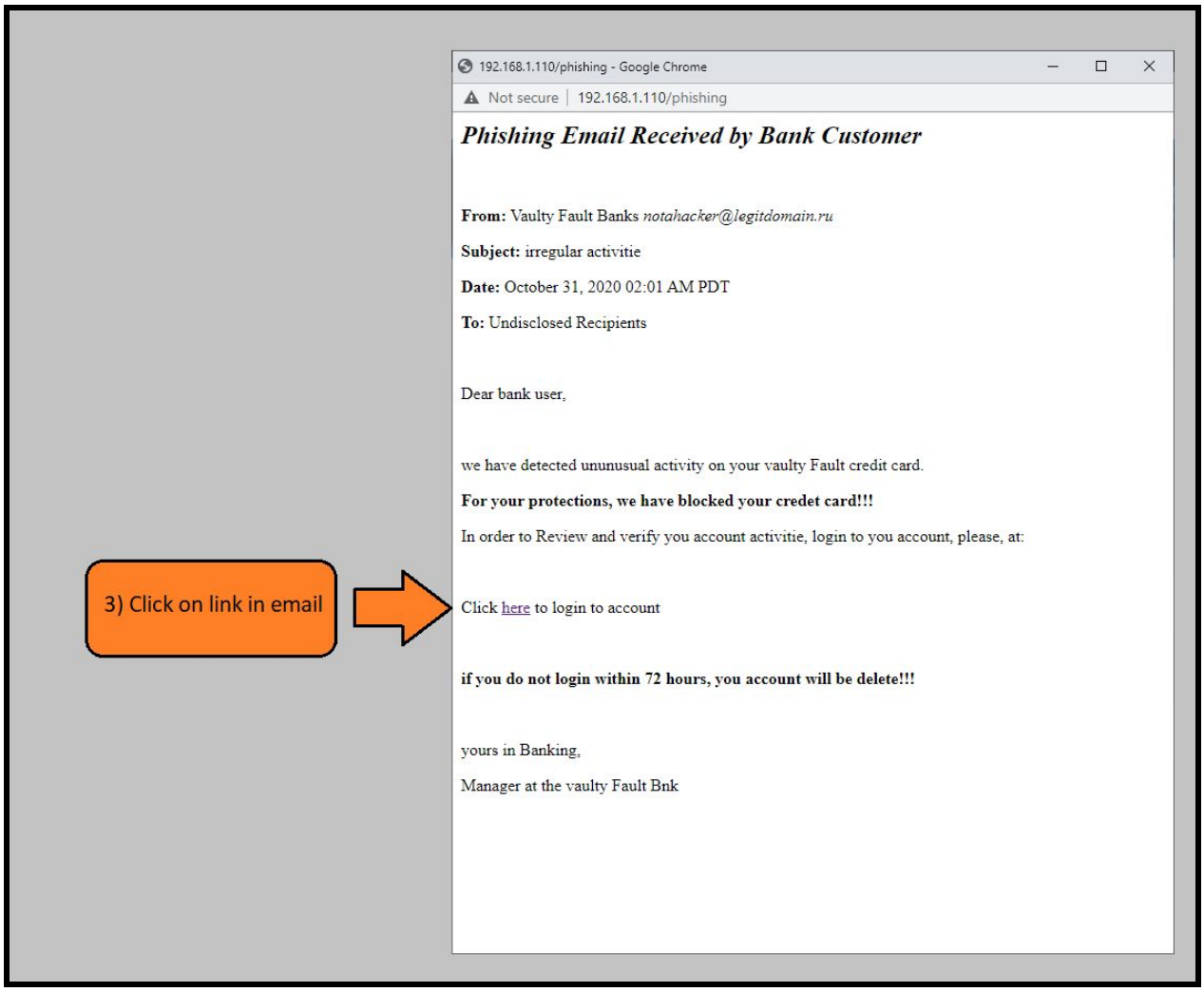
1. *Test whether site is vulnerable to a reflected XSS attack*

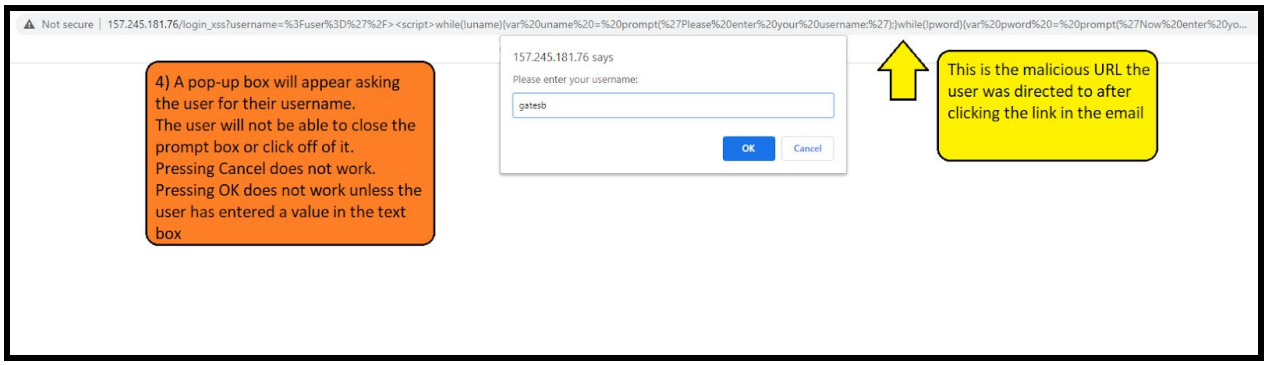


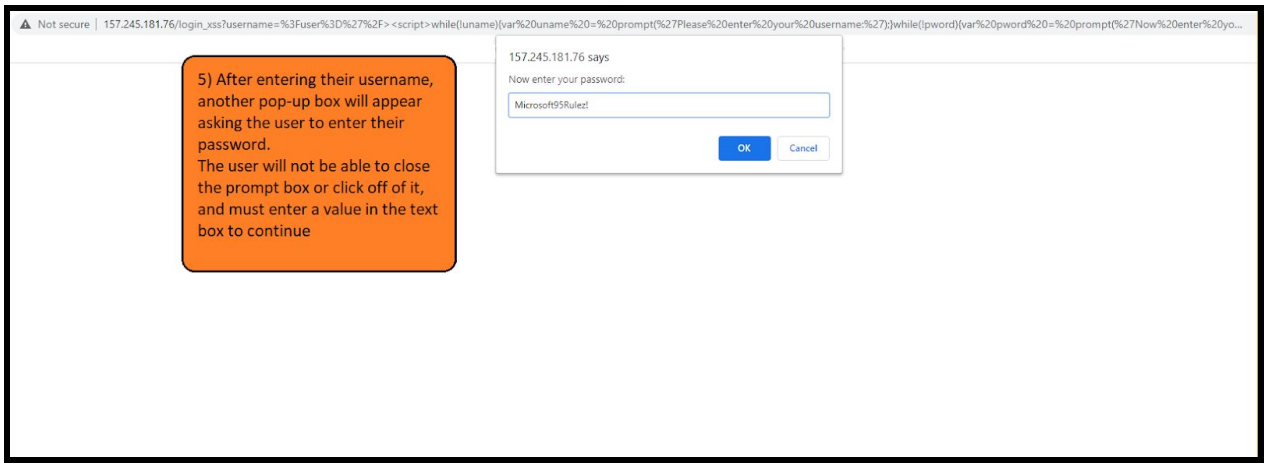
2. *Exploit the site's vulnerability to a reflected XSS attack via phishing*

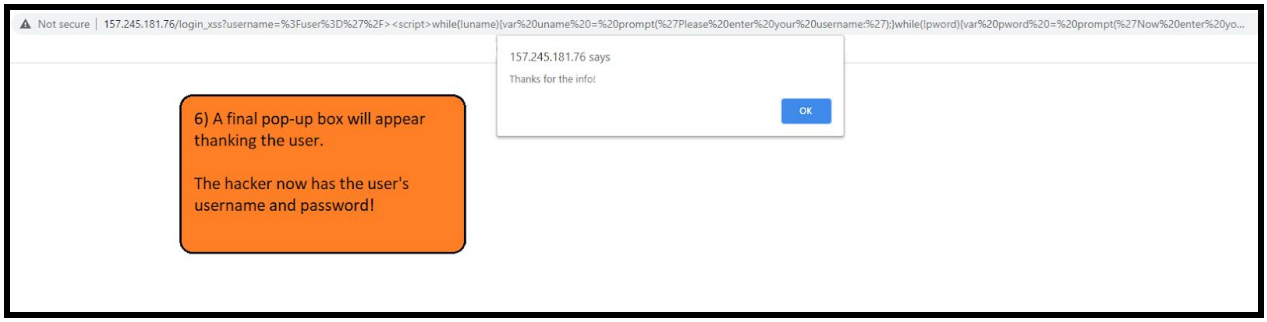





192.168.1.110/phishing - Google Chrome
Not secure | 192.168.1.110/phishing
Phishing Email Received by Bank Customer
From: Vaulty Fault Banks notahacker@legitdomain.ru
Subject: irregular activitie
Date: October 31, 2020 02:01 AM PDT
To: Undisclosed Recipients
Dear bank user,
we have detected ununusual activity on your vaulty Fault credit card.
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!!!
yours in Banking,
Manager at the vaulty Fault Bnk
3) Click on link in email

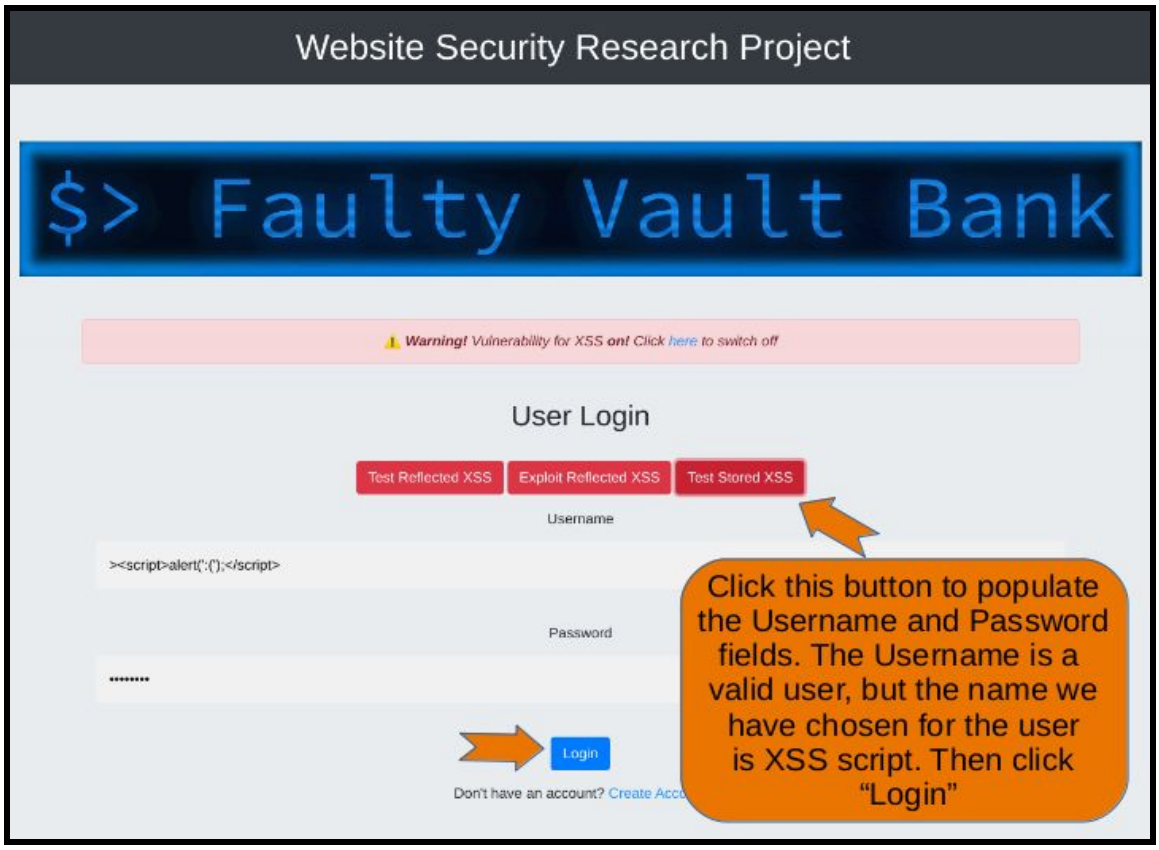

Not secure | 157.245.181.76/login_xss?username=%3Fuser%3D%27%2F><script>while(!uname){var%20uname%20=%20prompt(%27Please%20enter%20your%20username:%27);}while(!pword){var%20pword%20=%20prompt(%27Now%20enter%20yo...
157.245.181.76 says
Please enter your username:
gatesb
OK
Cancel
4) A pop-up box will appear asking the user for their username.
The user will not be able to close the prompt box or click off of it.
Pressing Cancel does not work.
Pressing OK does not work unless the user has entered a value in the text box
This is the malicious URL the user was directed to after clicking the link in the email

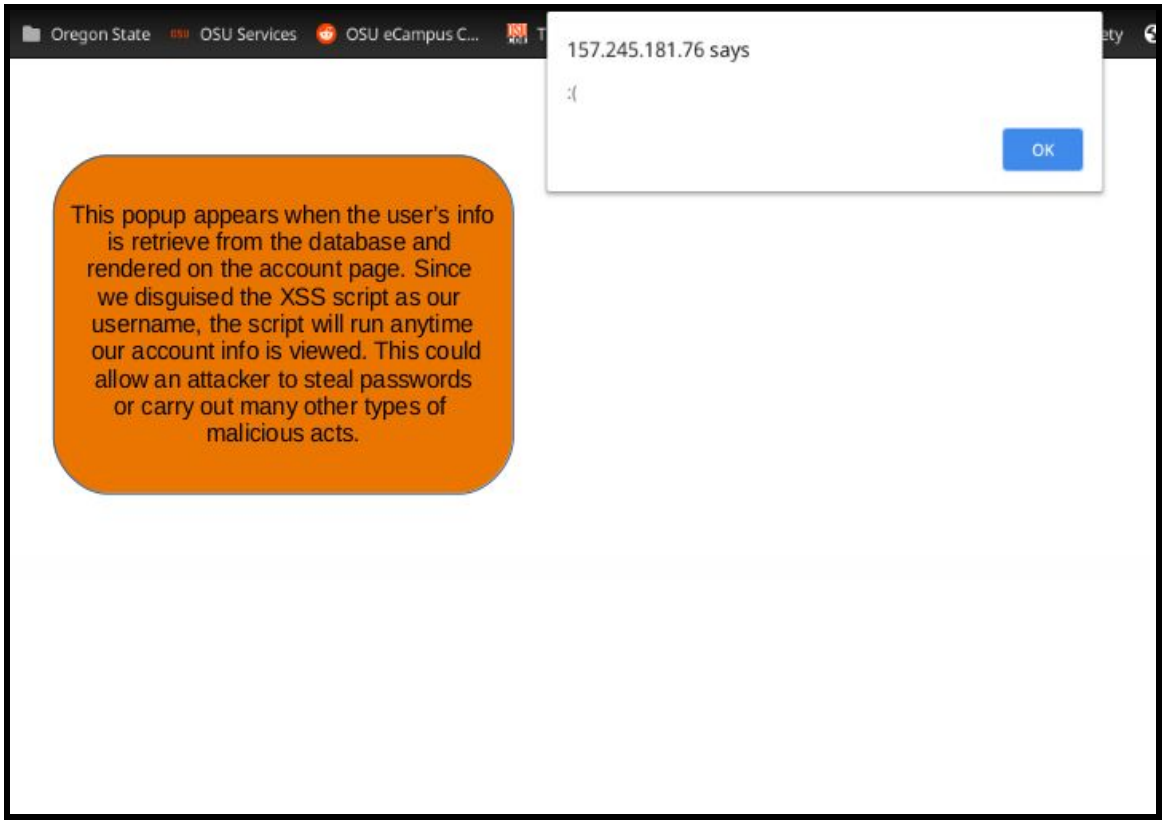




To view the hacker's page of stored usernames and passwords, visit:
http://157.245.181.76/hacker_info

Stored XSS


Website Security Research Project
$> Faulty Vault Bank
Warning! Vulnerability for XSS on! Click here to switch off
User Login
Test Reflected XSS
Exploit Reflected XSS
Test Stored XSS
Username
><script>alert(':(');</script>
Password
Login
Don't have an account? Create Acc
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"


Oregon State
OSU Services
OSU eCampus C...
157.245.181.76 says
:(
OK
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.

Website Hardening

To protect against xss attacks:

1. Login requests on a website should be sent via POST request rather than via GET request. In a POST request the username and password information is sent in the HTTP message body rather than in the URL where the user's login information would be clearly visible. This could be easily accessed via the browser's history or the link could be accidentally shared, exposing sensitive login information.

2. Ensure that all user input or untrusted data is validated and filtered in accordance with expected and valid input values. This can help prevent an XSS attack at the initial attempt.

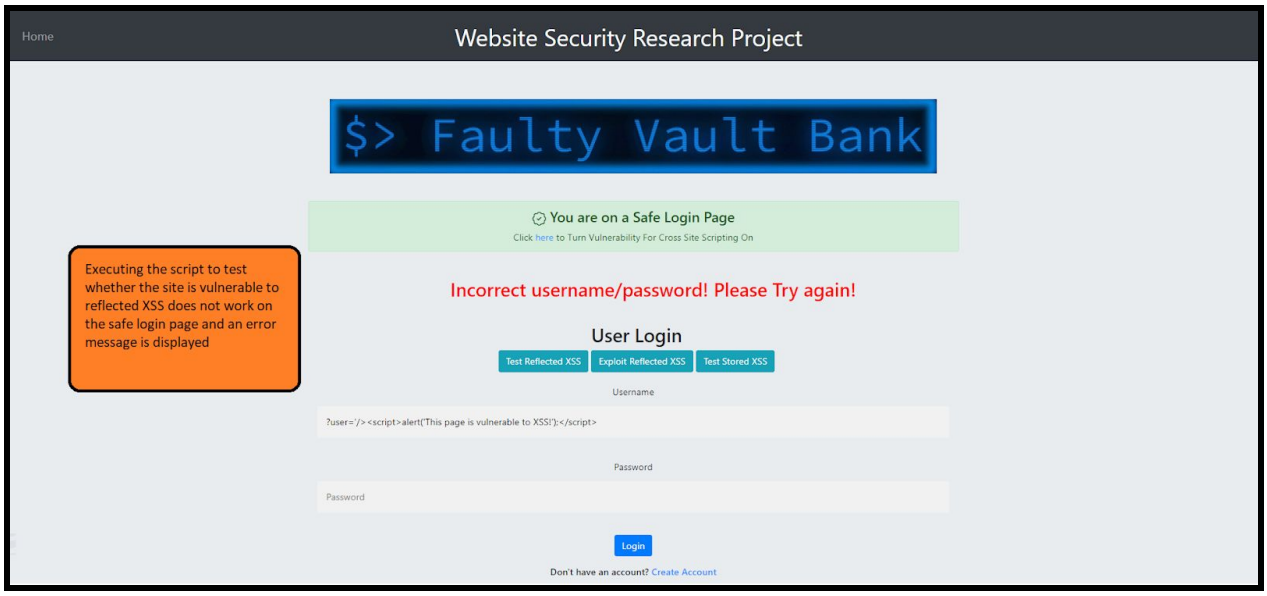
3. Make certain that all output is escaped. For instance, when inserting untrusted data into a HTML attribute, ensure that you do HTML escape. For example, '&' becomes '&', '<' becomes '&lt' and so on. In addition, check that you have enclosed the attribute in single (‘’) or double (“”) quotation marks. Never use backticks(`) to quote your attributes. Unquoted attributes can be easily broken out of using a number of different characters such as ';', '*', ',', '>', '%' and '+'.

4. Furthermore, it is important to be aware of any security limitations with your web template engine. Some web template engines such as Jinja2 can automatically escape HTML and prevent XSS attacks, however it is not able to protect against XSS attacks via attribute injection. As mentioned above, be sure to enclose all attributes in single or double quotes when utilizing Jinja expressions within them. Be sure that you are using the most up-to-date version of your web templating engine since the newest version may be patched against older security flaws.

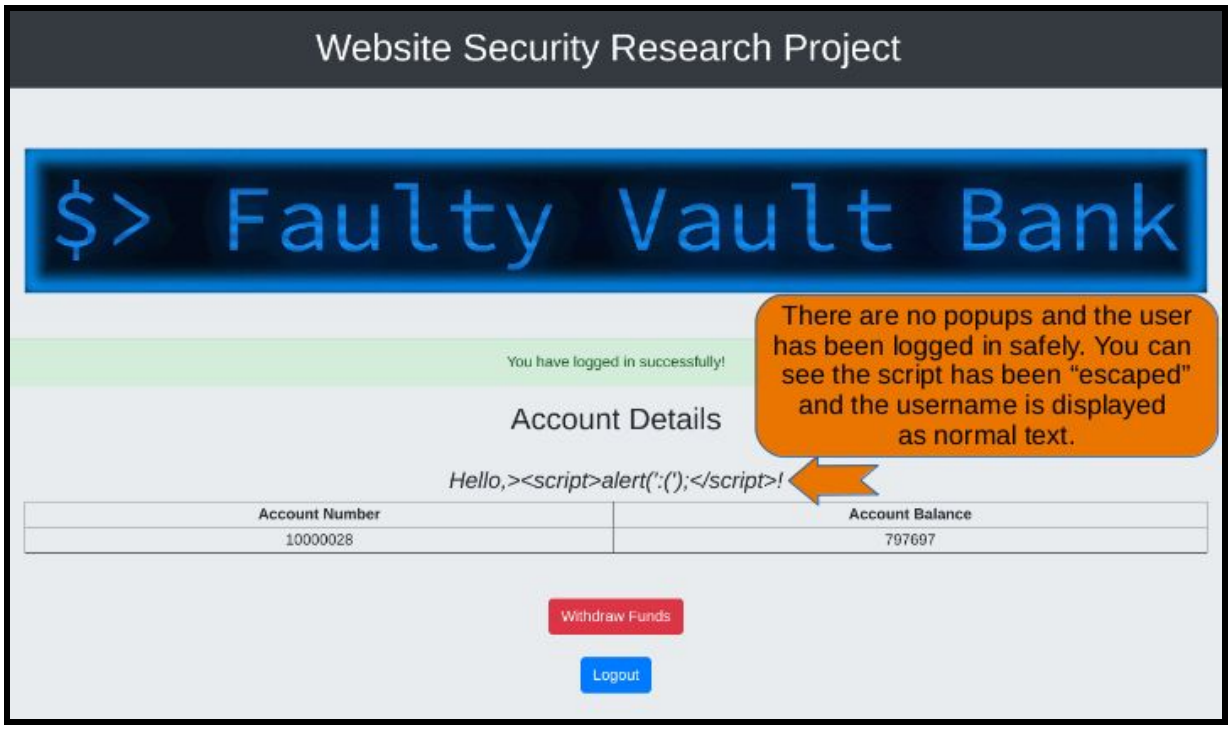
5. As a final line of defense, using a Content Security Policy (CSP) can reduce the severity of any XSS vulnerabilities that are still present in your site.

Screenshots showing the results of the XSS attacks against a secured website can be found on the next pages.

Attempt at Reflected XSS Attack After Website Hardening:



Attempt at Stored XSS Attack After Website Hardening:



Resources

https://flask.palletsprojects.com/en/1.1.x/security/

https://owasp.org/www-community/xss-filter-evasion-cheatsheet

https://owasp.org/www-community/Types_of_Cross-Site_Scripting

https://medium.com/@MichaelKoczwara/password-stealing-from-https-login-page-and-csrf-bypass-with-reflected-xss-76f56ebc4516

https://portswigger.net/web-security/cross-site-scripting

https://pentest-tools.com/blog/xss-attacks-practical-scenarios/

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