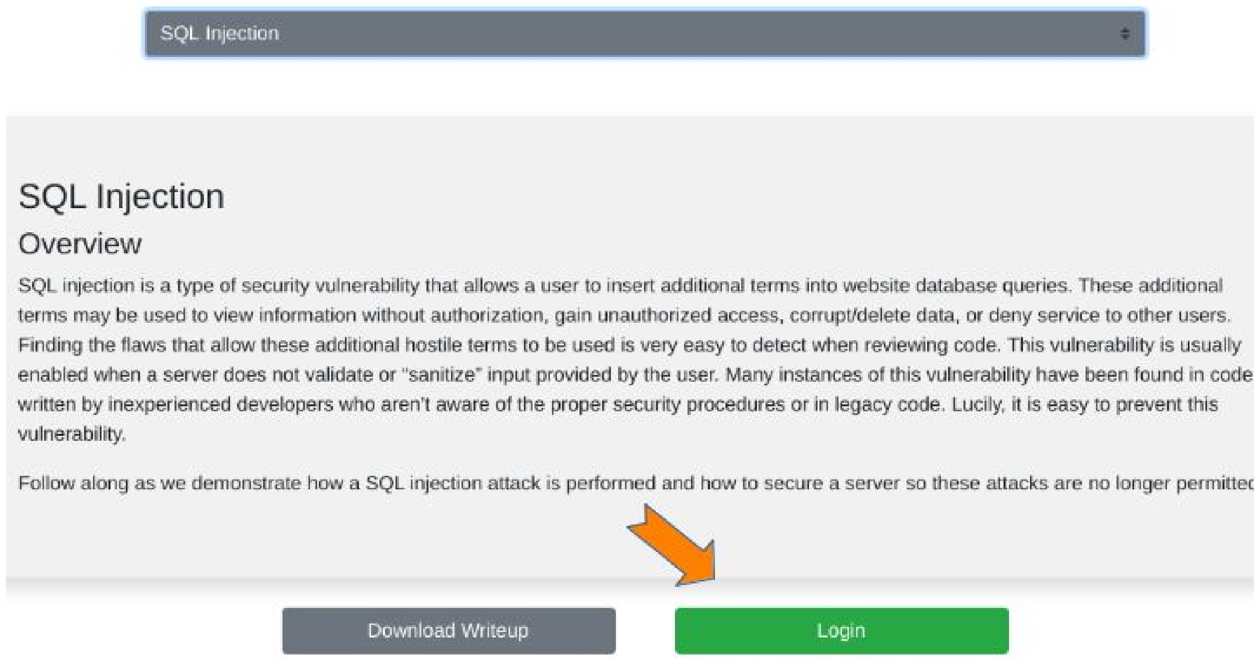
<u>SQL Injection</u>

Overview

SQL injection is a type of security vulnerability that allows a user to insert additional statements into website database queries. These statements may be used to view information without consent, gain unauthorized access, corrupt/delete data, or deny service to other users. Finding the flaws that allow these hostile statements 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 weren’t aware of the proper security procedures. Many times it is also found in legacy code. Lucily, it is easy to fix this vulnerability.

Attack Procedures



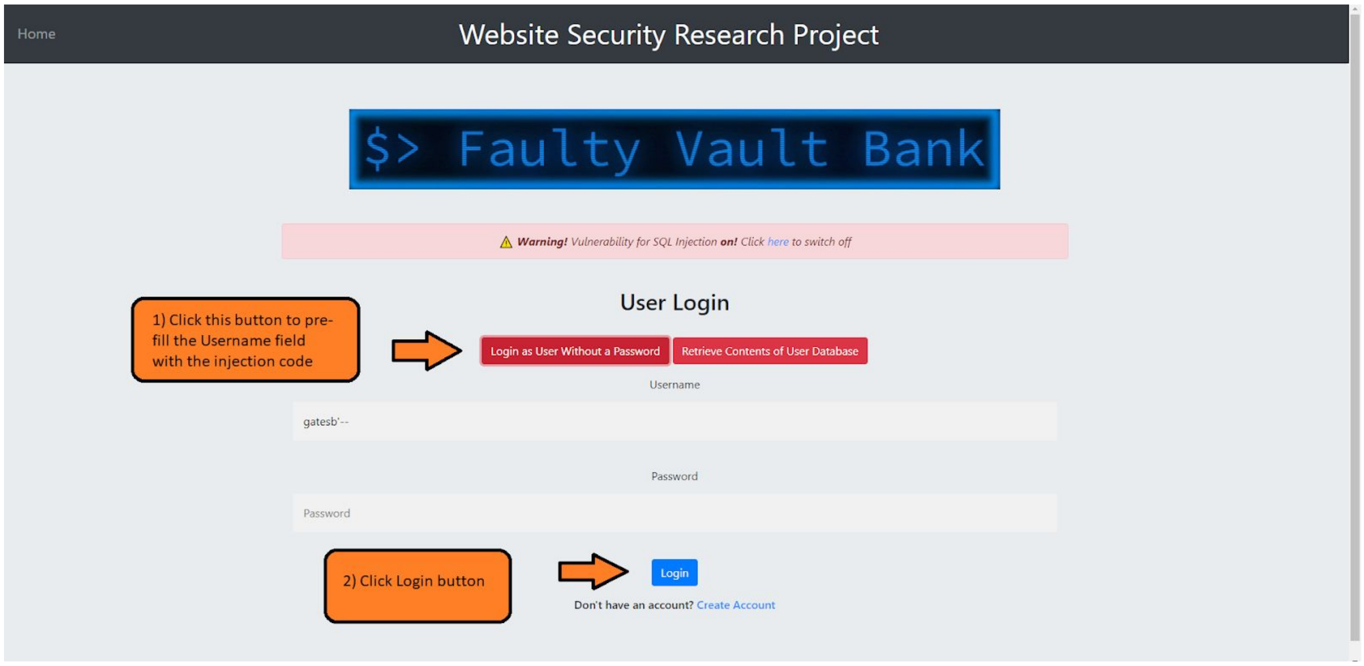
SQL Injection

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. Lucily, 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 permittec

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*

User Login

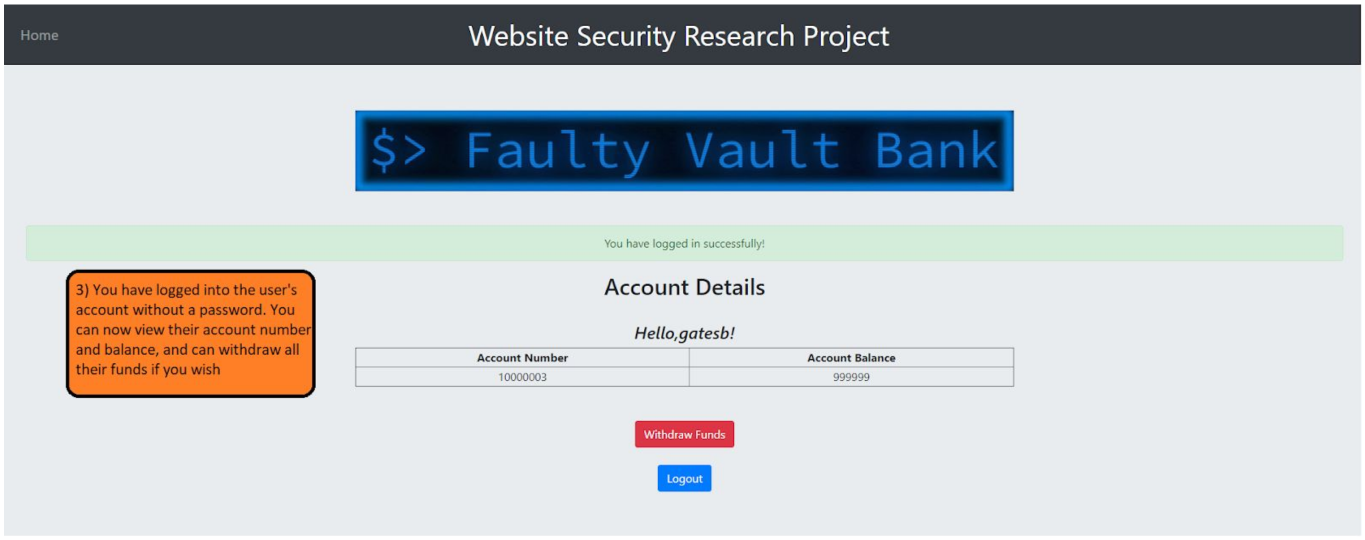
1) Click this button to pre-fill the Username field with the injection code

Login as User Without a Password
Retrieve Contents of User Database
Username

```
gatesb'--
```

2) Click Login button

Don't have an account? Create Account


Home

Website Security Research Project

$> Faulty Vault Bank

You have logged in successfully!

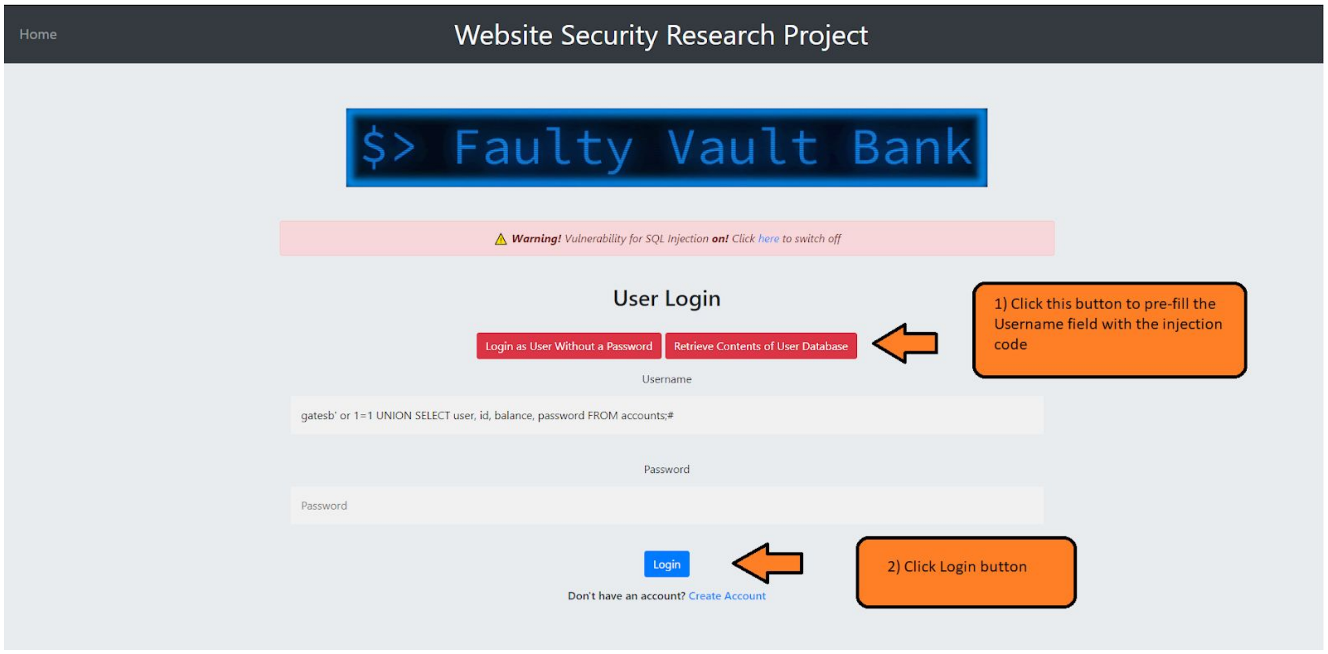
Account Details

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

Hello,gatesb!

Account Number	Account Balance
10000003	999999

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


Home
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

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

Username

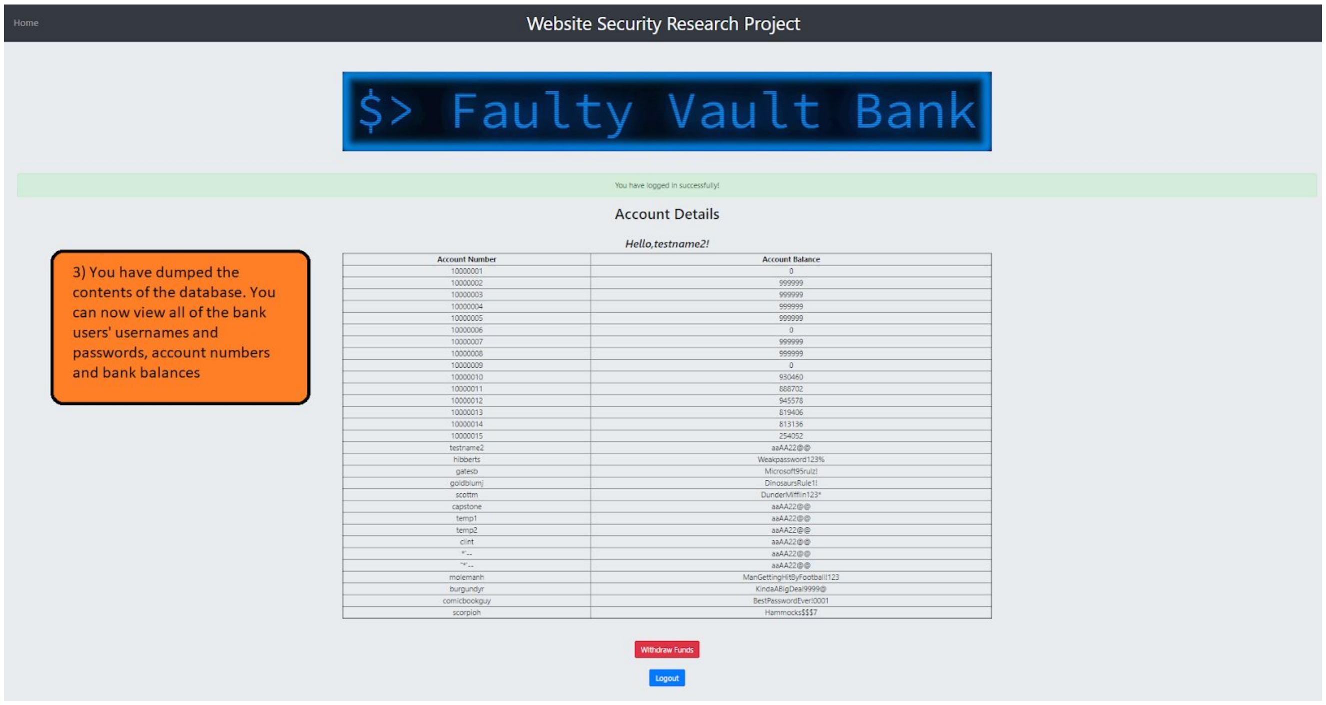
```
gatesb' or 1=1 UNION SELECT user, id, balance, password FROM accounts;#
```

Password

Password

2) Click Login button

Don't have an account? Create Account



You have logged in successfully!

Account Details

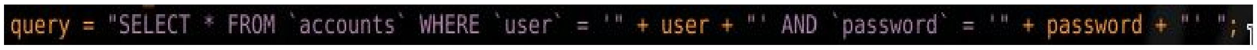
Hello,testname2!

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

Account Number	Account Balance
10000001	0
10000002	999999
10000003	999999
10000004	999999
10000005	999999
10000006	0
10000007	999999
10000008	999999
10000009	0
10000010	930460
10000011	888702
10000012	945578
10000013	819406
10000014	813136
10000015	254052
testname2	aaAA22@@
hibberts	Weakpassword123%
gatesb	Microsoft95rulz!
goldblumj	DinosaursRule1!
scottm	DunderMifflin123*
capstone	aaAA22@@
temp1	aaAA22@@
temp2	aaAA22@@
clint	aaAA22@@
*'--	aaAA22@@
'*'--	aaAA22@@
molemanh	ManGettingHitByFootball!123
burgundyr	KindaABigDeal9999@
comicbookguy	BestPasswordEver!0001
scorpioh	Hammocks$$$7

Securing Website

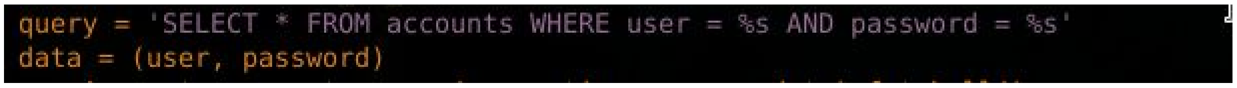
Websites are most vulnerable to SQL injection when input from the user is not validated or parameterized. This mistake is quite common and can be found all around the web. An example of user input not being parameterized can be seen in the image below.



```
query = "SELECT * FROM `accounts` WHERE `user` = '" + user + "' AND `password` = '" + password + "' ";
```

The construction of this query allows the user to “escape” the original statement and add additional statements of their own. In the example above, if a user entered ‘ OR 1=1;--
for the username when logging in, the query statement would be escaped and the user would be logged in without providing a username or password. If the user is able to insert their own statements like this, they can view sensitive information without consent or even cause data corruption and data loss. There are many harmful things that can happen if the user is not restricted from directly interacting with the database command interpreter, so we need to construct the query in a different manner.

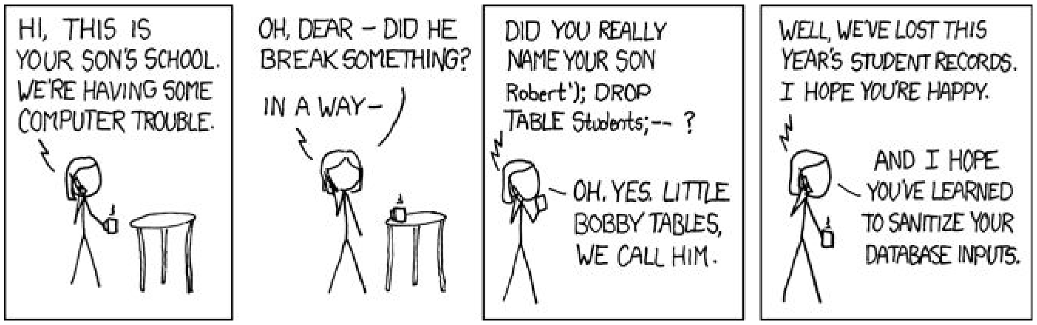
The approach we implemented to prevent SQL injection was using prepared statements and parameterized queries. An example of a parameterized query can be seen in the image below.



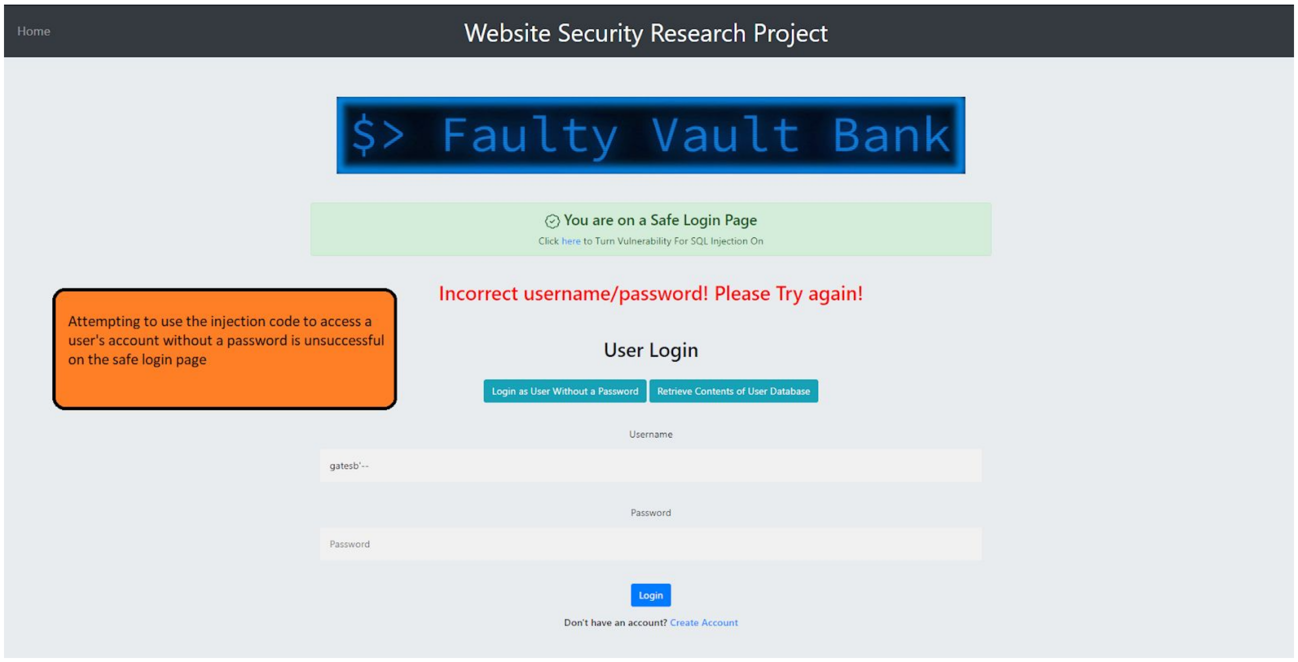
```
query = 'SELECT * FROM accounts WHERE user = %s AND password = %s'
data = (user, password)
```

When the query is constructed this way, the user is unable to escape the original query statement. If we take the example where the user entered ‘ OR 1=1;-- for the username, the statement would not be escaped in this case. Instead, the database would search for a record where user = “‘ OR 1=1;--”. This keeps the user from interacting directly with the interpreter.

***A funny example of what can happen if you don’t sanitize user input or use parameterized queries has been provided by https://xkcd.com/327/


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Here are some screenshots of what happens on the secure website when we try to inject the SQL that was successful on the vulnerable website.


Website Security Research Project

$> Faulty Vault Bank

You are on a Safe Login Page

Click here to Turn Vulnerability For SQL Injection On

Incorrect username/password! Please Try again!

Attempting to use the injection code to access a user's account without a password is unsuccessful on the safe login page

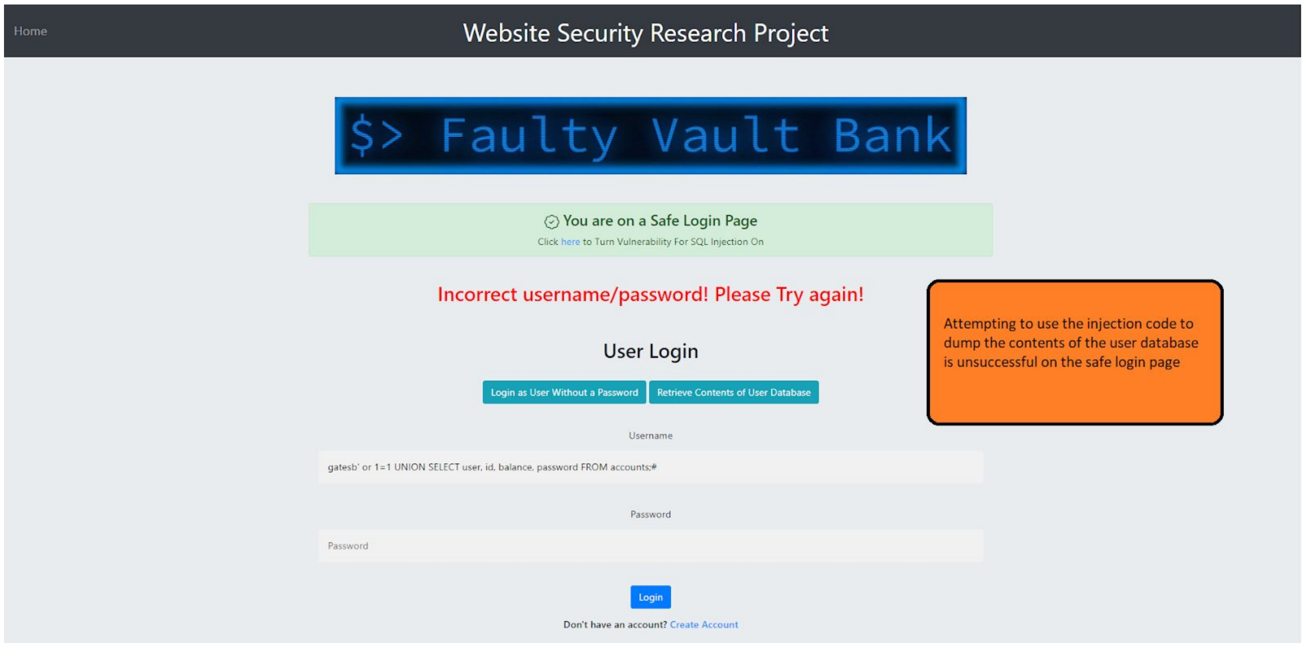
User Login

Username: gatesb'--

Password: Password

Don't have an account? Create Account

Home
Login as User Without a Password
Retrieve Contents of User Database
Login


Website Security Research Project

$> Faulty Vault Bank

You are on a Safe Login Page

Click here to Turn Vulnerability For SQL Injection On

Incorrect username/password! Please Try again!

Attempting to use the injection code to dump the contents of the user database is unsuccessful on the safe login page

User Login

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

Password: Password

Don't have an account? Create Account

Home
Login as User Without a Password
Retrieve Contents of User Database
Login

Resources

https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

https://portswigger.net/web-security/sql-injection

https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html#defense-option-1-prepared-statements-with-parameterized-queries