Web Application Security Audit

Final Report

Mandeep Parihar



Confidential

For Derek Yadlowski Only

Table of Contents

[Intent of Report 2](#_Toc89279677)

[Reconnaissance 2](#_Toc89279678)

[Attacks and Exploits 2](#_Toc89279679)

[General 2](#_Toc89279680)

[JavaScript 3](#_Toc89279681)

[Cross Site Request Forgery 4](#_Toc89279682)

[Cross Site Scripting 5](#_Toc89279683)

[SQL 7](#_Toc89279684)

[Invalidly Locking User Account 7](#_Toc89279685)

[Creating a New User Account 8](#_Toc89279686)

[Creating a New Bank Account 10](#_Toc89279687)

[Dropping Accounts 12](#_Toc89279688)

[Summation 13](#_Toc89279689)

# Intent of Report

This report is done to record security flaws, and corrections that can be made therein, of the itsc302finalproj SQL database, as well as the associated front-end to back-end files.

Various forms of testing were run, from JavaScript injections or SQL injections to cross site request forgery. All tests were malicious and aimed at causing a negative effect on the page, one that was not designed and should not be possible. Given the lax nature of the security posture, multiple attacks were able to cause damage to the pages, users, or database itself. All attacks used, the outcome from the attack, a description of it, and a potential solution to prevent future attacks of its kind will be provided in greater detail in a later section.

# Reconnaissance

To understand how to do these attacks, the files had to be explored. A cursory glance shows that there is a single SQL database file, multiple .jsp files for the front-end, and .java files to act as servlets for the front-end functionality going to the back-end values. Because the code can be viewed, it is possible to fashion attacks with known values, especially with access to the database. Full access to all users accounts and the admin accounts were valuable resources for exploitation. However, attacks can also be run from the index page itself, no login is required for those attacks. The attacks from the login page were, in fact, some of the worst attacks that could be done to the database itself. Since all of the attacks can be run the webpages directly, only NetBeans was required for a cross site request forgery.

# Attacks and Exploits

These attacks will be broken down into three parts: the first is the type of attack, the second is the way the attack was done, and the third is how that attack can be prevented. In instances where there is more than one attack for the type, the prevention methods will be bundled together into the third part.

## General

Type of Attack: Brute force

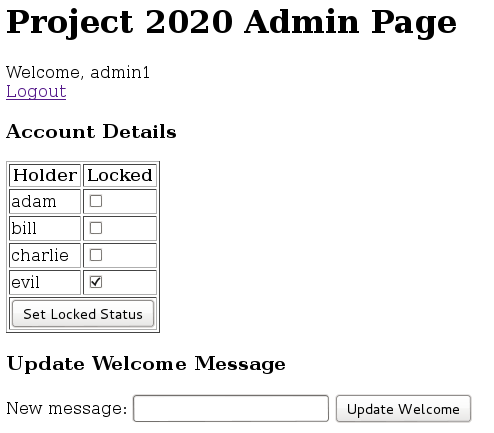
Procedure: A Python script could be made to run a rainbow table or even an easily available file like RockYou.txt against the login page. Due to the simple passwords used for the accounts, a brute force attack would likely find an accounts login credentials in little time. The script could also be designed to store the discovered username and password in a text file for quick reuse. Since no further credentials are required once logged in, a transfer could be done against the account holders will.

Steps to Remediation: The login page should have a limited number of attempts over a period of time. Hitting the limit will temporarily lock the user from further attempts until an elapsed period of time. This will slow the brute force attack down. Password requirements can also be enforced – making it so that users have to have a password of a certain length, containing specific types of characters, and does not contain their username within it. Lastly, two-factor authentication can be encouraged or outright enforced. Should a user use all of their login attempts, a warning message can be sent to the account owner, asking them to verify if they are the ones making the attempted logins. Additional actions can be taken from there, but that is beyond this documents scope.

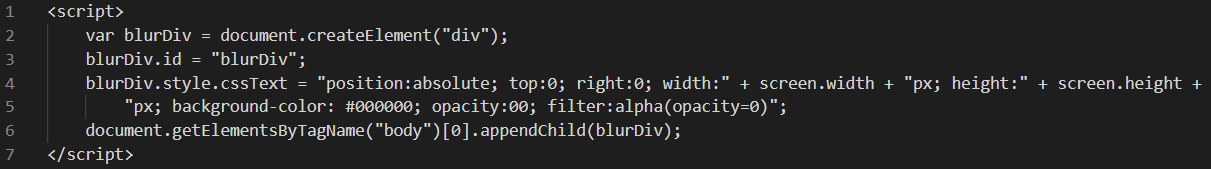
## JavaScript

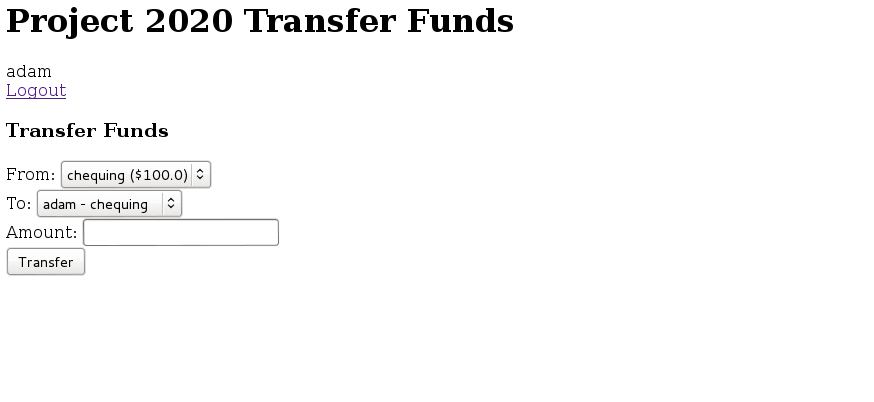
Type of Attack: CSS injection

Procedure: Completed through an admin account. Administrators have access to the “Welcome” message displayed to any logged in user on their bank page:



Entering and submitting the following string into the input field will initialize the attack:



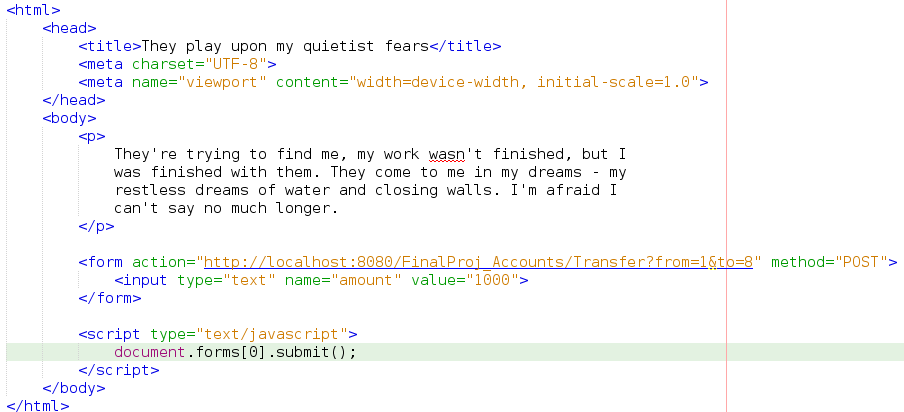
The above code will cause the entire page to be put behind an HTML DIV layer. This attack is persistent and will affect admins as well because they have a “Welcome” message on their login page. Because the code uses both **screen.width** and **screen.height***,* this attack will affect users with different resolutions for their screen. Additionally, because the color and opacity are set to 0, the page looks no different: The above screenshot was taken from an affected page. Comparing the color of the page to that of the admin page shows no difference. The “Welcome” message is displayed for all users, so anyone, including admins logging in will be affected by the injection.

Steps to Remediation: The input field where the injection was put should have a checker for script or HTML code. This can begin by checking for specific characters used for injections, such as “<” or “>”. Should special characters be entered, they can be changed out for something else, outright removed, or the update to the “Welcome” message can be denied.

## Cross Site Request Forgery

Type of Attack: Social engineering/cross site request forgery

Procedure: A victim needs to be logged into their bank page and the correct account ID for that users chequing or savings account needs to be used. From there, the victim needs to click on a malicious link that will forcibly redirect them to their bank page and make a transfer to a premade account for this attack. The code for the file is:



In the forms action, the “from” needs to be the victims account ID, and the “to” needs to be the malicious account. The amount can be changed to whatever is desired for the transfer. After running this attack, the $100 found in Adam’s chequing account went into the negatives:

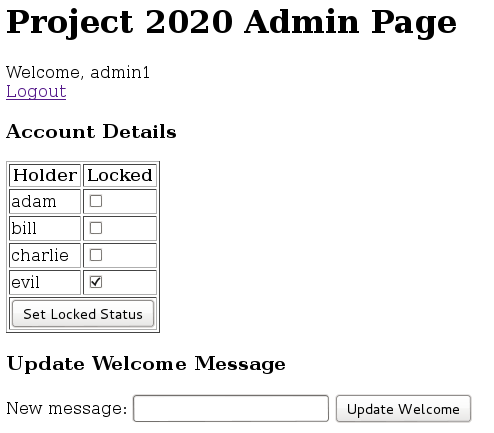


Steps to Remediation: A limit can be placed on transfers to prevent sending victims into negative values. This can either be on transferring more than they have or a hard limit for all transfers, but utilizing both will help. The best method to prevent this type of attack is to improve both employee and user security knowledge, and to teach them to not follow suspicious links in emails, texts, or on webpages. This attack can fashioned to target a specific individual if desired, making it dangerous for anyone who does not know they are being marked for it.

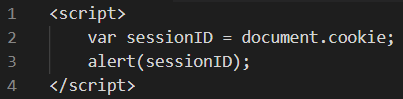
## Cross Site Scripting

Type of Attack: Browser side script

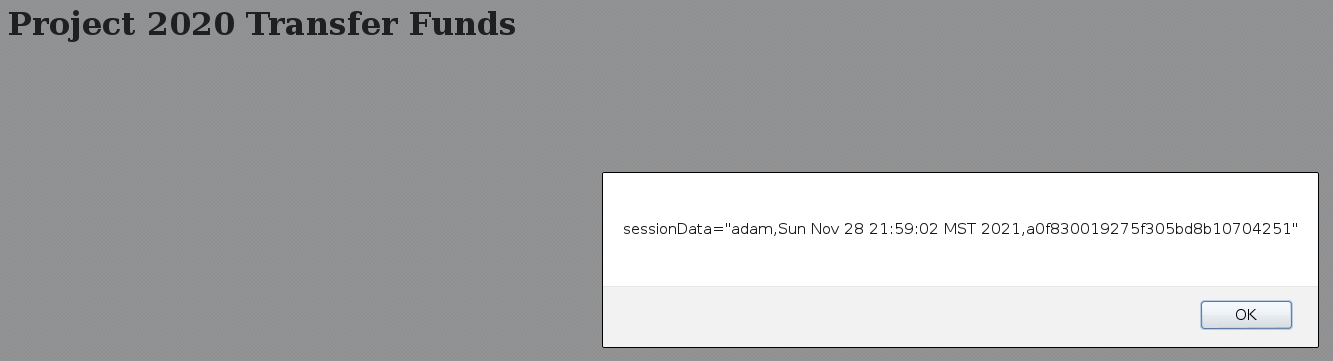
Procedure: Completed through an admin account. Administrators have access to the “Welcome” message displayed to any logged in user on their bank page:



Entering and submitting the following string into the input field will initialize the attack:



The above code will cause an alert to appear on any logged in users page displaying their session ID before loading most of the page. This attack is persistent until the “Welcome” message is changed to something else, but will also affect admins logging in, as their page also has a “Welcome” message:



Steps to Remediation: Improving security here is the same as before. Certain characters are used that are common in an attack but not common in a traditional welcome message. Unique symbols should be disallowed via regular expressions, or can be encoded differently, such as converting “<” into “&lt”. Words such as “script”, “style”, or “alert” can be blocked. Though “alert” is a common enough word, other words can be used instead, such as “attention”, while achieving the same result. Besides this, the nature of JavaScript means that it is case sensitive. Any string entered into the input field can be changed to be all uppercase or all lowercase. Lastly, any string entered can have double quotes appended to the beginning and end of it to try and nullify tags.

## SQL

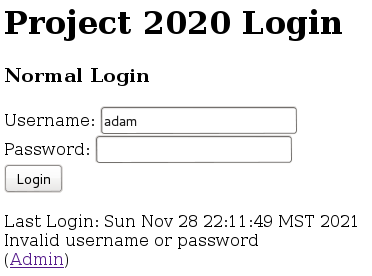
### Invalidly Locking User Account

Type of Attack: Denial of service

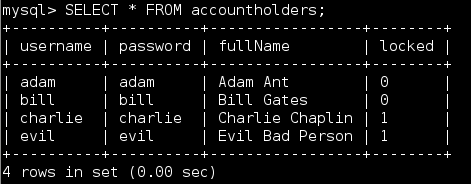
Procedure: This attack can be done on either the admin or normal user login page. Any string can be entered into the password field, but the following string will be entered into the username field:



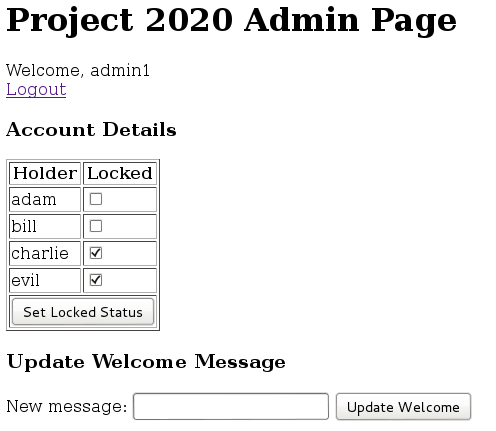
The above code will set the locked value to 1 for Charlie’s account. After attempting a login, the following error will be displayed on the login page:



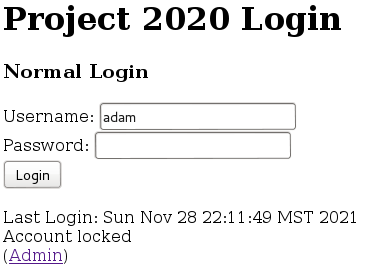
Commonly, a redirect to a blank page means the SQL attack failed, but this error means it succeeded. Checking the “accountholders” table in the database shows that the change took place:



Checking the admin page also shows the locked status has changed:



Attempting to login to Charlie’s account shows that it has been successfully locked:



Because this can be used to lock any account, a small modification to the query will unlock any previously locked account. With a pre-collected list of valid usernames for a brute force attack, any and all accounts can be locked. Locked accounts also return an error at the bottom of the screen that state that they are locked, which can inform an attacker which accounts to unlock with the credentials they just attempted a login with.

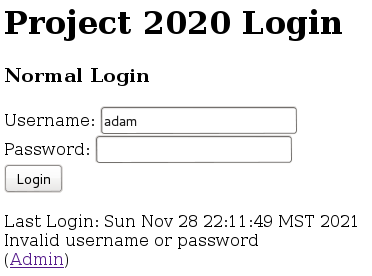
### Creating a New User Account

Type of Attack: Account injection

Procedure: This attack can be done on either the admin or normal user login page. Any string can be entered into the password field, but the following string will be entered into the username field:



The above code will create a new account in the database with the username “hacker” and the password “man”, and is not locked. After attempting a login, the following error will be displayed on the login page:



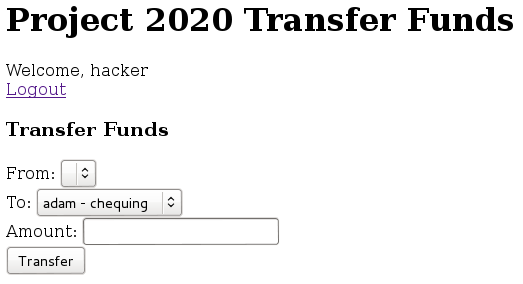
Commonly, a redirect to a blank page means the SQL attack failed, but this error means it succeeded. Checking the “accountholders” table in the database shows that the change took place:



Checking the admin page shows that the account is there:



The account can also be logged into:



This account can be used for transfers from normal users accounts into this one, as seen with the cross-site request forgery. Before that can happen, however, a bank account will need to be made for this account.

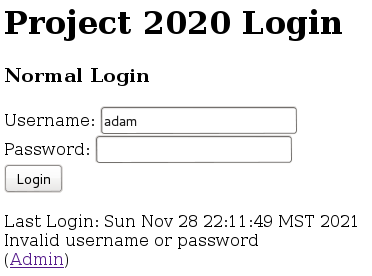
### Creating a New Bank Account

Type of Attack: Account injection

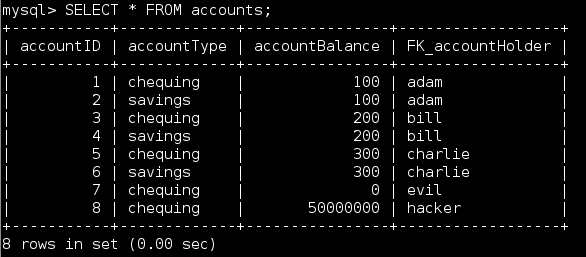
Procedure: This attack can be done on either the admin or normal user login page. Any string can be entered into the password field, but the following string will be entered into the username field:



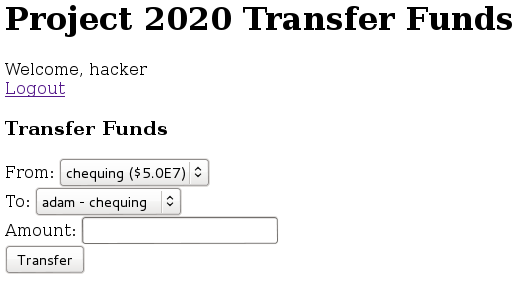
The above code will create a new chequing account in the database under the “hacker” account, with a balance of $50 million. After attempting a login, the following error will be displayed on the login page:



Commonly, a redirect to a blank page means the SQL attack failed, but this error means it succeeded. Checking the “accounts” table in the database shows that the change took place:



Logging into the “hacker” account once more shows a valid chequing account with the $50 million balance:



With such an egregious amount of money in the account, this is an obvious problem. Since accounts can be affected, it should be possible to make further negative changes to existing accounts and account holders.

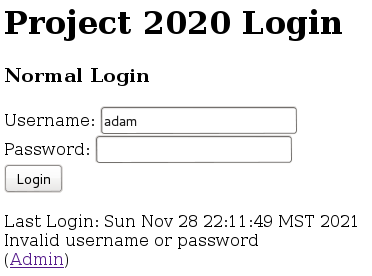
### Dropping Accounts

Type of Attack: Account removal

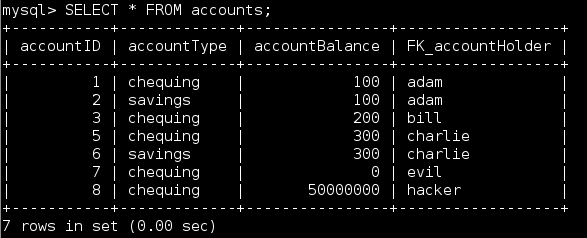
Procedure: This attack can be done on either the admin or normal user login page. Any string can be entered into the password field, but the following string will be entered into the username field:



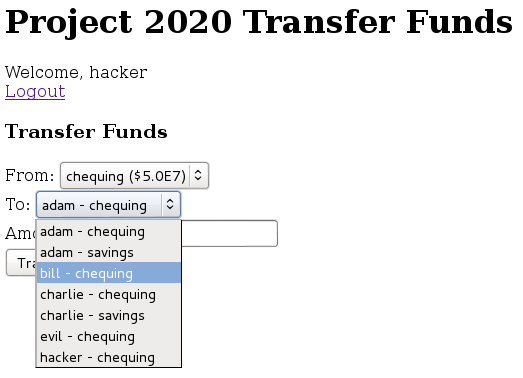
The above code will drop Bill’s entire savings account. After attempting a login, the following error will be displayed on the login page:



Commonly, a redirect to a blank page means the SQL attack failed, but this error means it succeeded. Checking the “accounts” table in the database shows that the change took place:



Logging into an account shows that Bill’s savings account is no longer an option:



With not simple ways to restore the database from a previous point, and no history of changes, the account is lost permanently and will need to be remade entirely again.

Steps to Remediation: All of the SQL attacks functioned the same way, it was only their outcome that was altered. Knowing this, it is possible to make changes that would prevent one attack that will also prevent all of them. Making quotation marks return exceptions will stop things like the account entry. Do not allow multi queries, this is how for the abuse of terminating an existing query to login and making it into something malicious. Admin accounts should only be given specific privileges so only one is able to lock accounts, whereas a lower one can change the “Welcome” message. Privileges should be more selectively given. Ultimately, placeholders in queries should be used. These prepared statements are faster than building a list of acceptable query code, and will determine what is and is not valid. These placeholders can be used to prevent a large amount of SQL attacks and injections.

# Summation

Multiple types of attacks are possible on the webpage and database, but there is overlap between how some of them work. For this reason, securing the webpages and database becomes easier. Certain words or characters commonly found in injections and SQL attacks can be invalidated by alternate encodings for the character, using regular expressions to prevent their use, blocking them outright, or all characters can be converted to all uppercase or lowercase, which is especially effective against JavaScript. Putting any string entered in any input field within double quotes can also defuse multiple attack types. Cross site request forgeries are more unique in that their prevention needs to come from informing and improving employee and user security understanding. Malicious links are disguised, so teaching people to think twice about clicking on something that is suspicious is the best defense against them. Placeholders in servlet queries can protect against most SQL attacks that come in. They will check if something is valid or not and handle a lot of work for backend developers. Applying all of the above will prevent all of the attacks that were used in this report, in addition to other that were not discovered.