Annex C: Individual Project Report

Student's Name	Muhammad Ennaayattulla	**Admin No**	192026S
Course	Diploma in Cyber Security and Digital Forensics	**Acad / Sem**	2020S1
Module Name	IT2555 Applications Security Project	**Module Group**	3
Module Supervisor	Ms Verawaty		
Team Leader	Siti Sarah Binte Sa'ad Bagarib	**Team No**	1

Instructions:

(i) Each leader/member shall submit one form to Blackboard by Week 18 Wed 5pm.

1. Project Description

For this project, our team was tasked to create two versions of a website. A secure and vulnerable version.

Our team planned to create a website that would be like a forum/blog, similar to Reddit. The website has both user accounts and admin accounts.

- Users can choose from multiple topics to discuss. They can read other people's posts, comment or upvote them, reply to a comment or make their own posts.

- The website would also have administrators to moderate it. The administrators can create topics and receive feedback from users, among other functions. They would also have the authority and privilege of deleting posts and/or terminating user accounts whenever necessary.

Our team was tasked to exploit the 2 vulnerabilities that we chose from the OWASP Top 10 Web vulnerabilities in the vulnerable version of the website, while the secure version should protect against chosen vulnerabilities. The team members and our chosen vulnerabilities are as follows:

- Sarah Bagarib – Broken Authentication, Security Misconfiguration
- Ko Jia Ling – Broken Access Control, Injection
- Muhammad Ennaayattulla – Cross Site Scripting, Sensitive Data Exposure

Our team was also motivated to add in additional security features and mitigate more than 2 vulnerabilities in the secure version of the website.

2. Individual Task

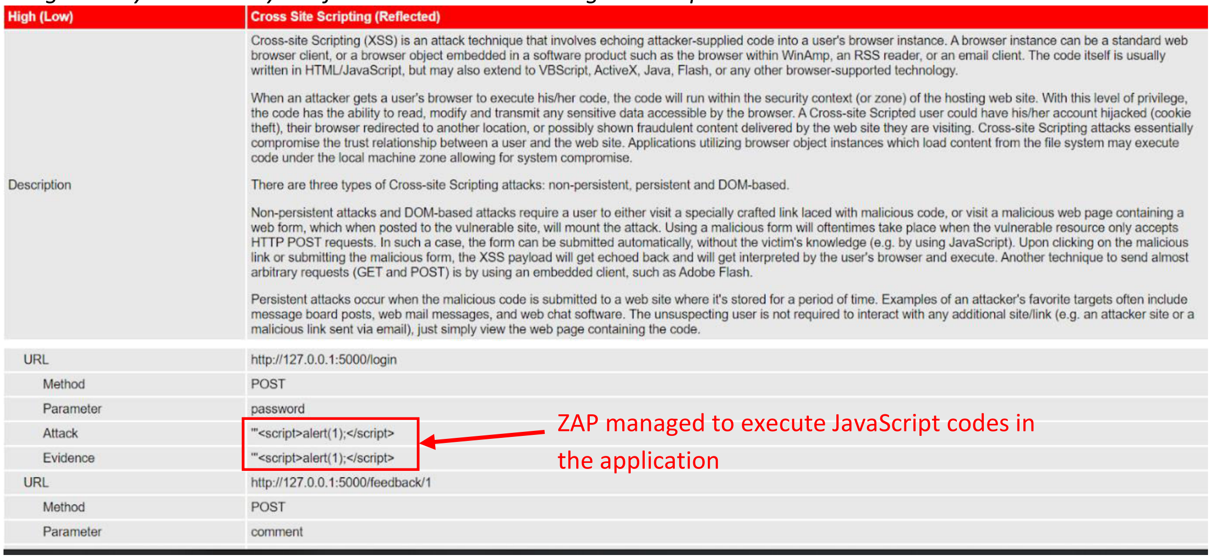
The 2 OWASP Top 10 Web vulnerabilities that I have chosen are **Cross Site Scripting** and **Sensitive Data Exposure**. Additionally, I added some general security features and implemented secure practices that protect against other vulnerabilities, some of which may not be in OWASP Top 10. The following documents the implementation of my vulnerability in the vulnerable version of the website, as well as my solution for mitigation of the vulnerability in the secure version of the website.

Cross Site Scripting

Cross-site scripting (XSS) describes a web security vulnerability that allows attackers to compromise user interactions by inserting malicious scripts designed to hijack vulnerable applications.

During the dynamic scan of the vulnerable version of the application, the scanning tool ZAP was able to identify the risk of XSS.

Image 1.0 Dynamic Analysis of vulnerable version using ZAP - report



High (Low)	Cross Site Scripting (Reflected)
Description	Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology. When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based. Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash. Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.
URL	http://127.0.0.1:5000/login
Method	POST
Parameter	password
Attack	'"<script>alert(1);</script>
Evidence	'"<script>alert(1);</script>
URL	http://127.0.0.1:5000/feedback/1
Method	POST
Parameter	comment

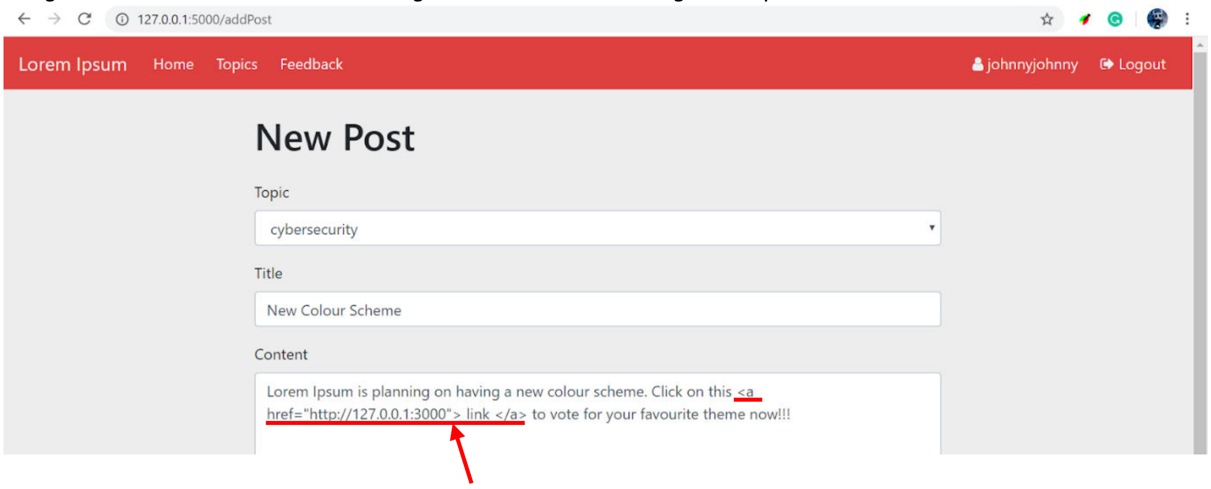
Implementation: Inserting malicious code into application via input fields

Any input field can be a prime target for XSS. If user input is not validated or escaped properly, attackers may be able to execute arbitrary HTML and JavaScript in the victim's browser.

In the following example, an attacker attempts to redirect users to malicious websites by inserting HTML codes into an input field.

Step 1. Attacker inserts an anchor tag that will redirect the user to the malicious website when they click on the word "link".

Image 1.1 Attacker includes anchor tag on "link" when creating a new post



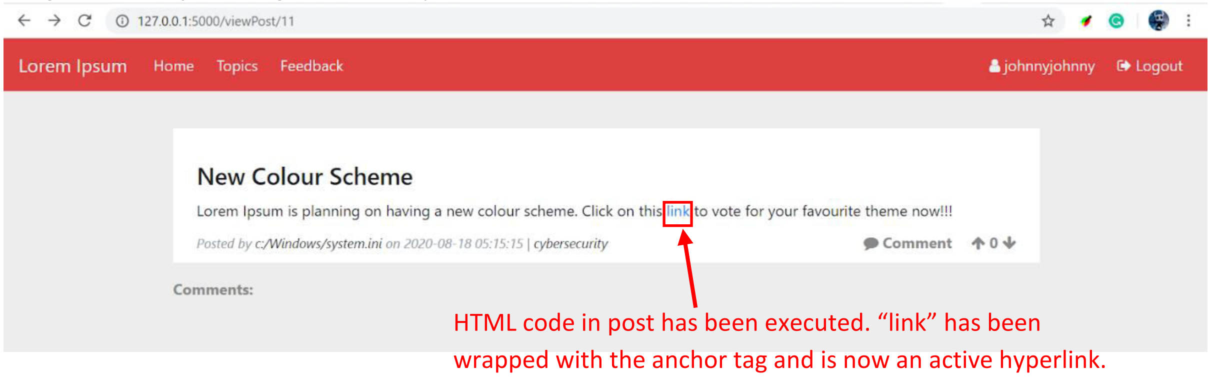
New Colour Scheme

Lorem Ipsum is planning on having a new colour scheme. Click on this <a href="http://127.0.0.1:3000"> link </a> to vote for your favourite theme now!!!

Anchor tag with href attribute set to malicious website link.

Step 2. Attacker submits the post with the HTML code. The post is stored together with the HTML code into the application's database. When the post is retrieved from the database to be displayed, the application does not escape the HTML codes and instead executes the code. This causes the word "link" to be an active hyperlink, which directs users to the malicious website. This type of XSS is called *Stored XSS.*

Image 1.2 Post after being submitted by the attacker.



New Colour Scheme

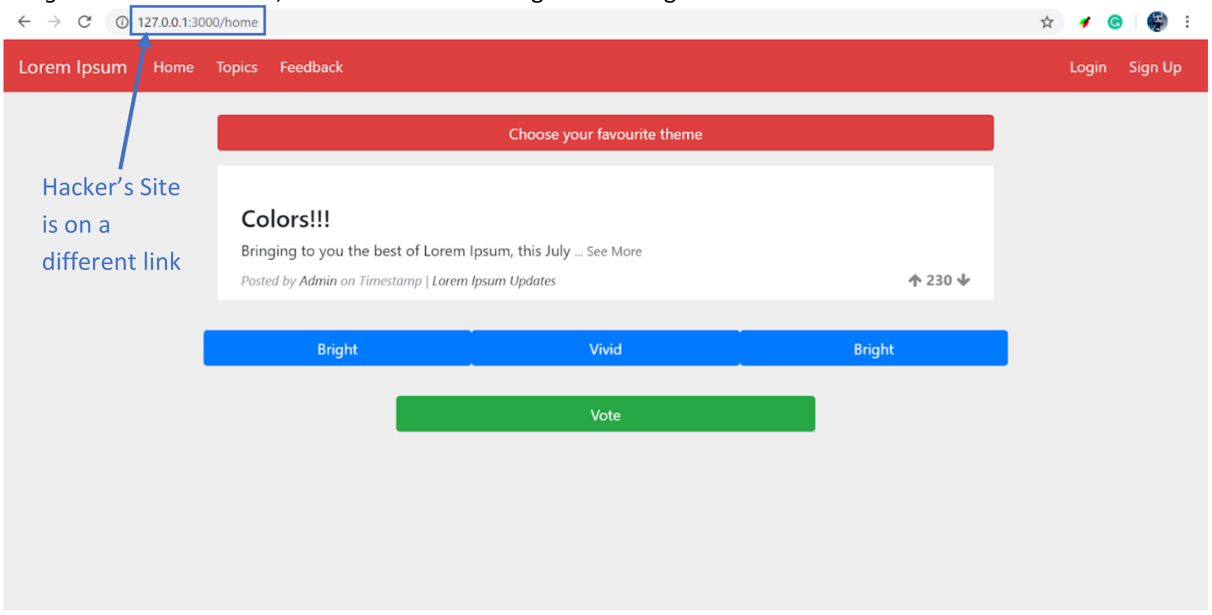
Lorem Ipsum is planning on having a new colour scheme. Click on this link to vote for your favourite theme now!!!

Posted by c:/Windows/system.ini on 2020-08-18 05:15:15 | cybersecurity

HTML code in post has been executed. "link" has been wrapped with the anchor tag and is now an active hyperlink.

So, when an unsuspecting user clicks on the link, they will get redirected to the malicious website.

Image 1.2.1 Hacker's Site, which mirrors the design of the original website.


127.0.0.1:3000/home
Lorem Ipsum
Home
Topics
Feedback
Login
Sign Up
Hacker's Site is on a different link
Choose your favourite theme
Colors!!!
Bringing to you the best of Lorem Ipsum, this July ... See More
Posted by Admin on Timestamp | Lorem Ipsum Updates
230
Bright
Vivid
Bright
Vote

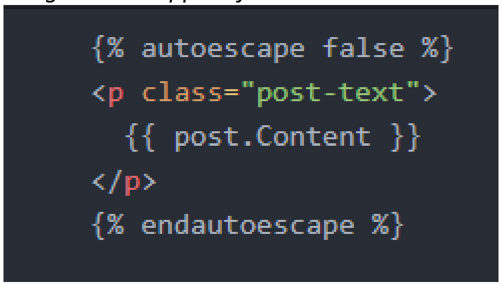
The user may have already lost important information such as session information just by clicking on the link and visiting the site. Moreover, since the Hacker's site looks similar to the original website, the user may unintentionally leak more important personal information which can lead to many negative consequences.

Solution: Escaping and validating all user inputs properly

Escaping and validating all user inputs would prevent the application from executing scripts and other code that may have been included in the input.

The Python Flask framework automatically escapes user input. Even if an attacker were to include malicious scripts or code into their inputs, Flask would safely escape the characters and treat them as normal input instead of executing them as code. In order to implement the XSS vulnerability, I had to manually disable Flask's autoescape feature in the vulnerable version of the website.

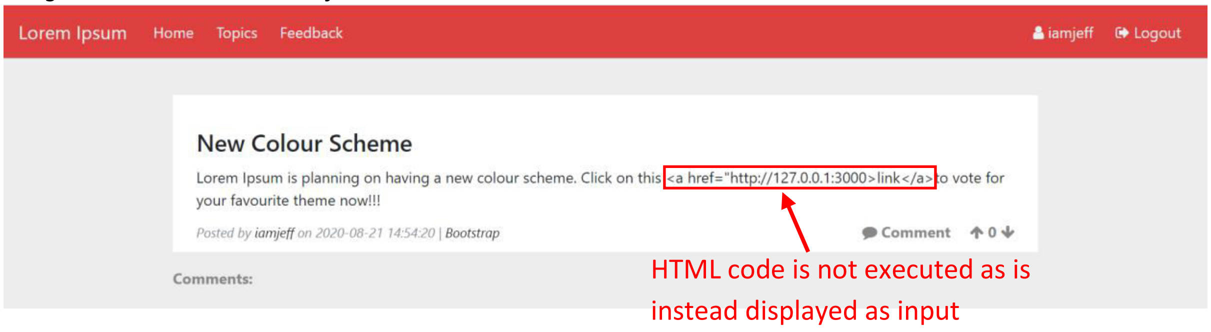
Image 1.3.1 Snippet of code in where autoescape has been manually disabled (vulnerable version)



```
{% autoescape false %}
<p class="post-text">
  {{ post.Content }}
</p>
{% endautoescape %}
```

In the following picture, the same post is created in the secure website. However, the application managed to safely escape the input, causing the HTML codes in the post to not be executed. It is instead treated as part of the post content.

Image 1.3.2 Post with HTML after it has been submitted on secure version



New Colour Scheme

Lorem Ipsum is planning on having a new colour scheme. Click on this <a href="http://127.0.0.1:3000>link</a> to vote for your favourite theme now!!!

Posted by iamjeff on 2020-08-21 14:54:20 | Bootstrap

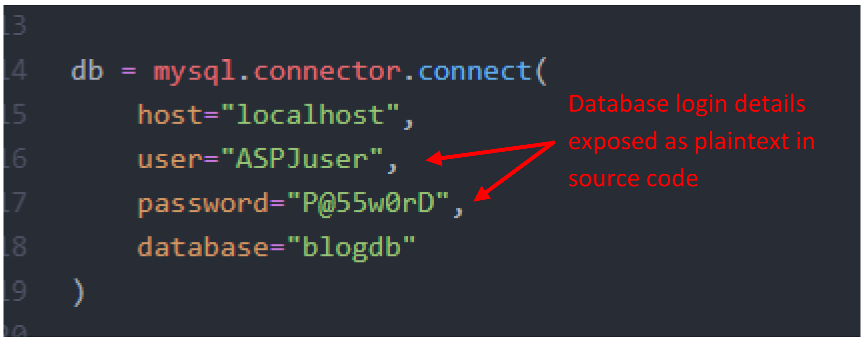
HTML code is not executed as is instead displayed as input

Sensitive Data Exposure

Sensitive Data Exposure vulnerability occurs when an application fails to adequately protect sensitive information, leaving it open to accidental exposure or hacking.

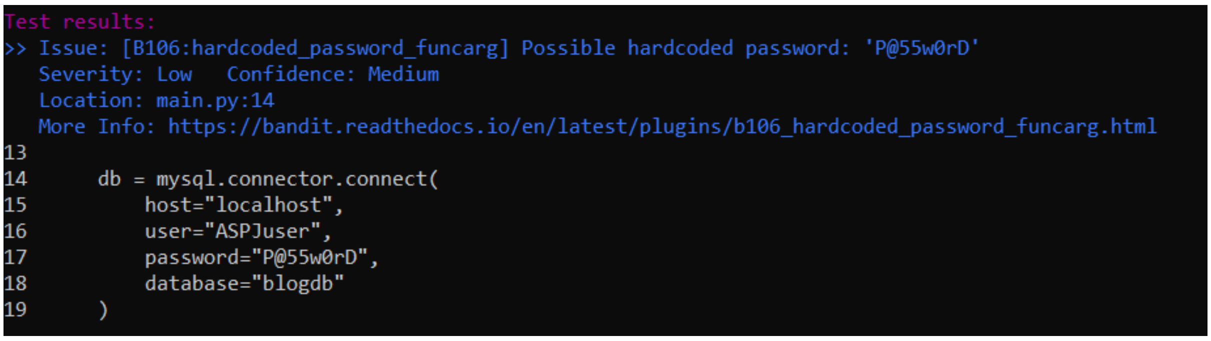
During the static scanning of the vulnerable application, the static scanning tool Bandit managed to identify an instance of Sensitive Data Exposure.

Image 2.0.1 Snippet of source code where database connection is initialized (vulnerable version)



```
db = mysql.connector.connect(
    host="localhost",
    user="ASPJuser",
    password="P@55w0rD",
    database="blogdb"
)
```

Image 2.0.2 Static Analysis of vulnerable version using Bandit - Report



```
Test results:
>> Issue: [B106:hardcoded_password_funcarg] Possible hardcoded password: 'P@55w0rD'
   Severity: Low   Confidence: Medium
   Location: main.py:14
   More Info: https://bandit.readthedocs.io/en/latest/plugins/b106_hardcoded_password_funcarg.html
13
14      db = mysql.connector.connect(
15          host="localhost",
16          user="ASPJuser",
17          password="P@55w0rD",
18          database="blogdb"
19      )
```

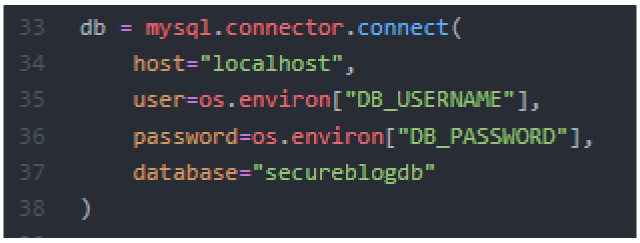
Implementation 1: Database credentials are visible in code

Database credentials should not be visible in code as anyone with access to the source code may use it to carry out unauthorized creation, modification or deletion of data in the database. It is also generally a good practice to keep all important information, such as credentials, as confidential as possible. Image 2.0.1 shows the implementation of this vulnerability in the vulnerable version source code.

Solution 1: Set important values as Environment Variables

Setting environment variables allows for crucial but sensitive values, such as database credentials, to be used in the source code without being explicitly shown in plaintext. This can be done with the via the Python os module. After setting the environment variables, the os module can refer to the value in the variable, given the variable's name.

Image 2.1.1 Snippet of source code with Environment variables (secure version)



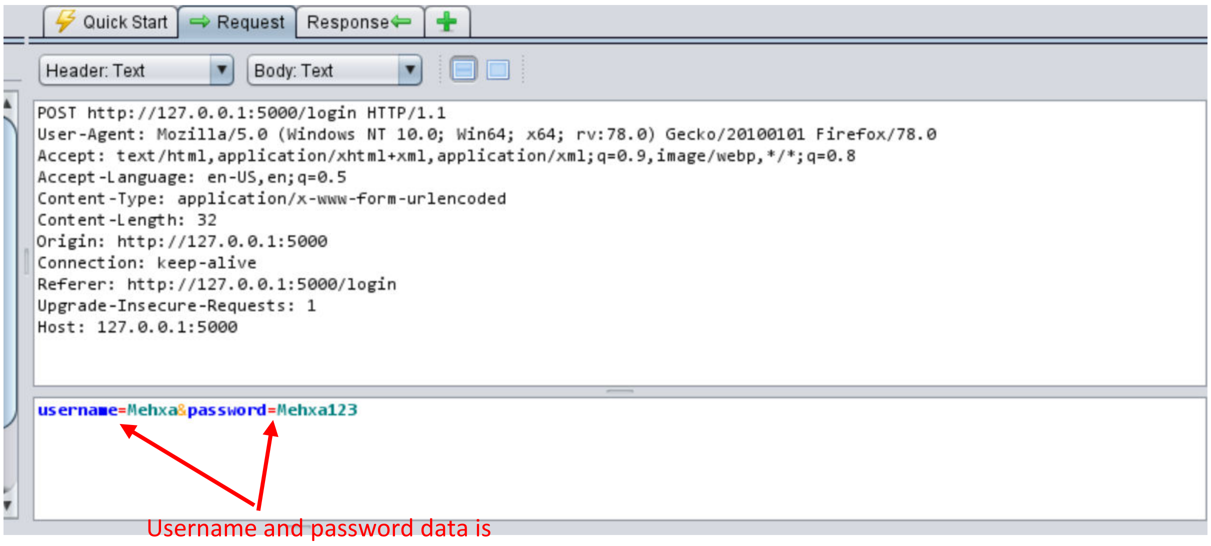
```
db = mysql.connector.connect(
    host="localhost",
    user=os.environ["DB_USERNAME"],
    password=os.environ["DB_PASSWORD"],
    database="secureblogdb"
)
```

Implementation 2: All data transferred is not encrypted and is in plaintext

Sensitive and important information such as usernames and passwords should be encrypted in order to protect against MITM(Man-In-The-Middle) attacks. An attacker may use a sniffing tool to sniff incoming and outgoing data from the application. Unencrypted data can be easily available to attackers in this manner.

The following image shows how unencrypted data can be picked up by attackers using the tool, ZAP.

Image 2.2.1 Username and Password data sniffed by ZAP



```
POST http://127.0.0.1:5000/login HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Origin: http://127.0.0.1:5000
Connection: keep-alive
Referer: http://127.0.0.1:5000/login
Upgrade-Insecure-Requests: 1
Host: 127.0.0.1:5000
```

```
username=Mehxa&password=Mehxa123
```

Username and password data is captured in plaintext when a user tries to login

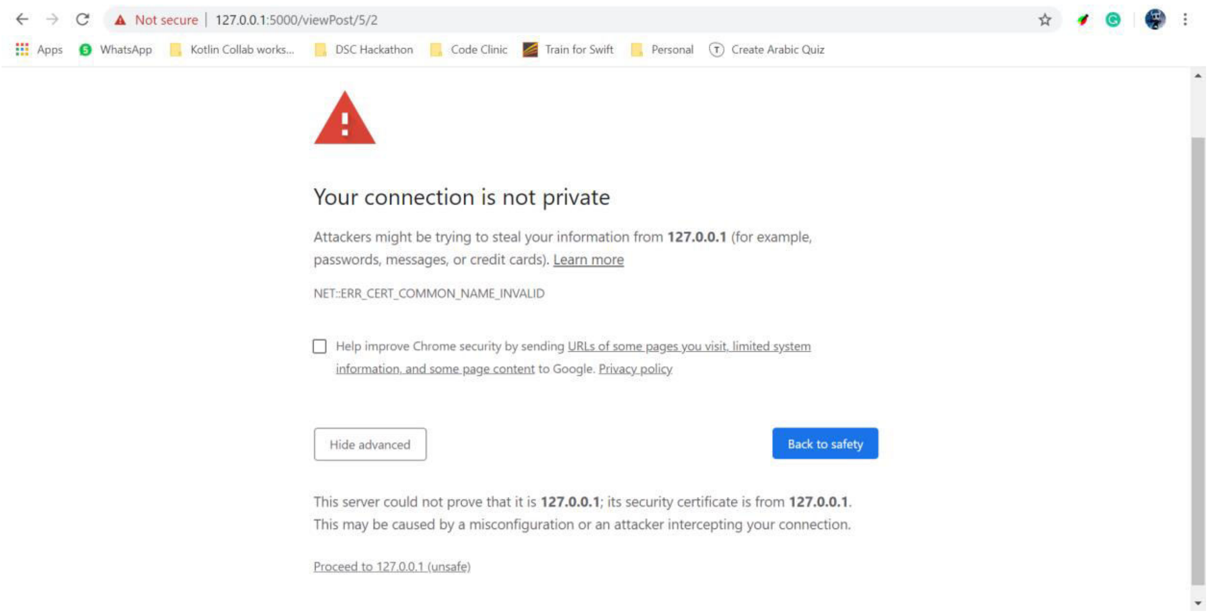
Attackers can easily use this exposed sensitive data to carry out MITM or Replay attacks.

Solution 2.1: Implement HTTPS

HTTPS (Hypertext Transfer Protocol Secure) makes use of SSL (Secure Sockets Layer)/TLS (Transport Layer Security) to securely encrypt all communication between server and client. This will ensure that even if the attacker manages to sniff the sensitive data, it would be encrypted and the attacker would not be able to do anything with it.

For this solution, I had to use a self-signed certificate using OpenSSL. SSL certificates issued by trusted Certificate Authorities must be paid for. Using a certificate changes the URL of the website from http to https. However, the following error will appear.

Image 2.2.2.1 Error is raised by browser as self-signed certificate is not recognised/authorised



Your connection is not private

Attackers might be trying to steal your information from **127.0.0.1** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_COMMON_NAME_INVALID

Help improve Chrome security by sending URLs of some pages you visit, limited system information, and some page content to Google. Privacy policy

This server could not prove that it is **127.0.0.1**; its security certificate is from **127.0.0.1**. This may be caused by a misconfiguration or an attacker intercepting your connection.

Proceed to 127.0.0.1 (unsafe)

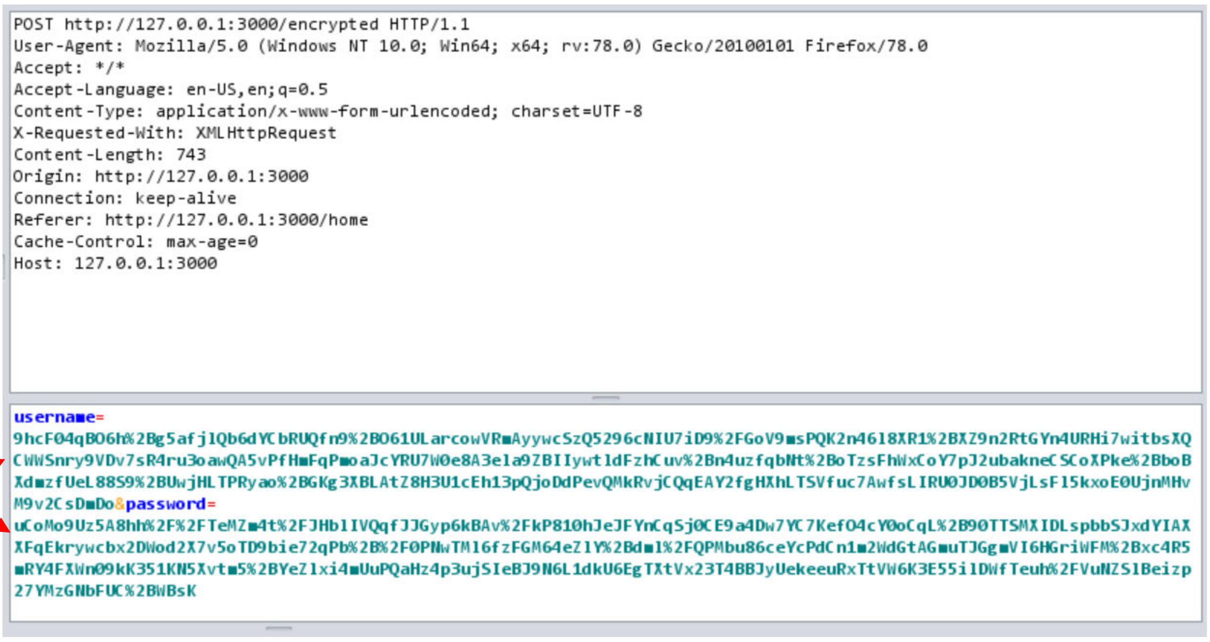
Even if the certificate was configured with the correct name and added my computer's Trusted Root Certificates, an error, ERR_CERT_AUTHORITY_INVALID, would be raised as the organisation that signed the cert, myself, is not recognised by Google. This error is unavoidable when using self-signed certs and can only be resolved by purchasing a certificate from a Certificate Authority that Google recognises. This solution is documented as a proof of concept as it cannot be implemented without costs.

Solution 2.2: Use Strong Encryption Algorithms to encrypt data

Strong encryption algorithms can be implemented to encrypt the data before it is being transferred so that it can be transferred safely to its destination without being exposed or modified by attackers. The secure version of the website implemented RSA (Rivest-Shamir-Adleman) encryption using a JavaScript library called JSEncrypt. This form of encryption implements a PKI(Public Key Infrastructure); meaning that a public and private key pair would be generated and used for encryption.

The following image shows encrypted data sniffed by ZAP.

Image 2.2.2.2 Encrypted data sniffed by ZAP

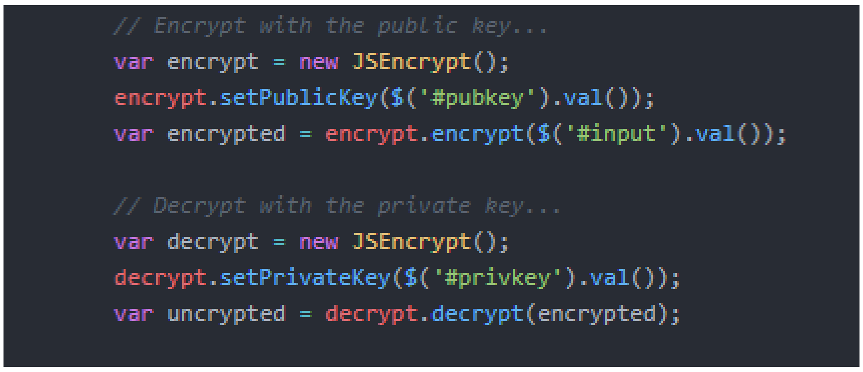


```
POST http://127.0.0.1:3000/encrypted HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 743
Origin: http://127.0.0.1:3000
Connection: keep-alive
Referer: http://127.0.0.1:3000/home
Cache-Control: max-age=0
Host: 127.0.0.1:3000
```

```
username=
9hcF04qBO6h%2Bg5afjlQb6dYCbRUQfn9%2BO61ULarcowVRmAyywcSzQ5296cNIU7iD9%2FGoV9msPQK2n4618XR1%2BXZ9n2RtGYn4URHi7witbsXQ
CWWSnry9VDv7sR4ru3oawQA5vPfHmFqPmoaJcYRU7W0e8A3ela9ZBIIywtldFzhCuv%2Bn4uzfqbNt%2BoTzsFhWxCoY7pJ2ubakneCSCoXPke%2BboB
XdmzfUeL88S9%2BUwjHLTPRyao%2BGKg3XBLAtZ8H3U1cEh13pQjoDdPevQMkRvjCQqEAY2fgHXhLTSVfuc7AwfsLIRU0JD0B5VjLsF15kxoE0UjnMHv
M9v2CsDmDo&password=
uCoMo9Uz5A8hh%2F%2FTeMZm4t%2FJHblIVQqfJJGyp6kBAv%2FkP810hJeJFYnCqSj0CE9a4Dw7YC7KefO4cY0oCqL%2B90TTSMXIDLspbbSJxdYIAX
XFqEkrywcbx2DWod2X7v5oTD9bie72qPb%2B%2F0PNwTM16fzFGM64eZ1Y%2Bdm1%2FQPMbu86ceYcPdCn1m2WdGtAGmuTJGgmVI6HGriWFM%2Bxc4R5
mRY4FXWn09kK351KN5Xvtm5%2BYeZ1xi4mUuPQaHz4p3ujSIeBJ9N6L1dkU6EgTXtVx23T4BBJyUekeeuRxTtVW6K3E55i1DWfTeuh%2FVuNZS1Beizp
27YMzGNbFUC%2BWBsK
```

Username and password have been encrypted

Image 2.2.2.3 Snippet of source code with JSEncrypt



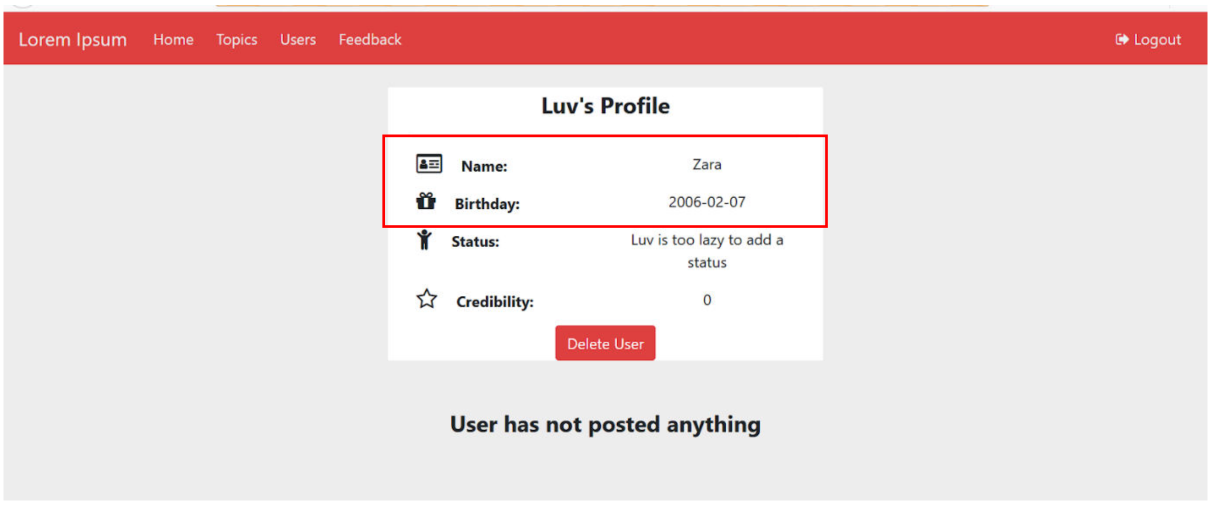
```
        // Encrypt with the public key...
        var encrypt = new JSEncrypt();
        encrypt.setPublicKey($('#pubkey').val());
        var encrypted = encrypt.encrypt($('#input').val());

        // Decrypt with the private key...
        var decrypt = new JSEncrypt();
        decrypt.setPrivateKey($('#privkey').val());
        var uncrypted = decrypt.decrypt(encrypted);
```

Implementation 3: Sensitive information such as name and birthday are unnecessarily displayed

Displaying personal data where it is unnecessary is also a form of sensitive data exposure. It is a good security practice to only display the necessary data regarding a person.

The following example shows personal information such as name and birthday being displayed.

Image 2.3.1 Profile page on vulnerable website



Luv's Profile

Name:	Zara
Birthday:	2006-02-07
Status:	Luv is too lazy to add a status
Credibility:	0

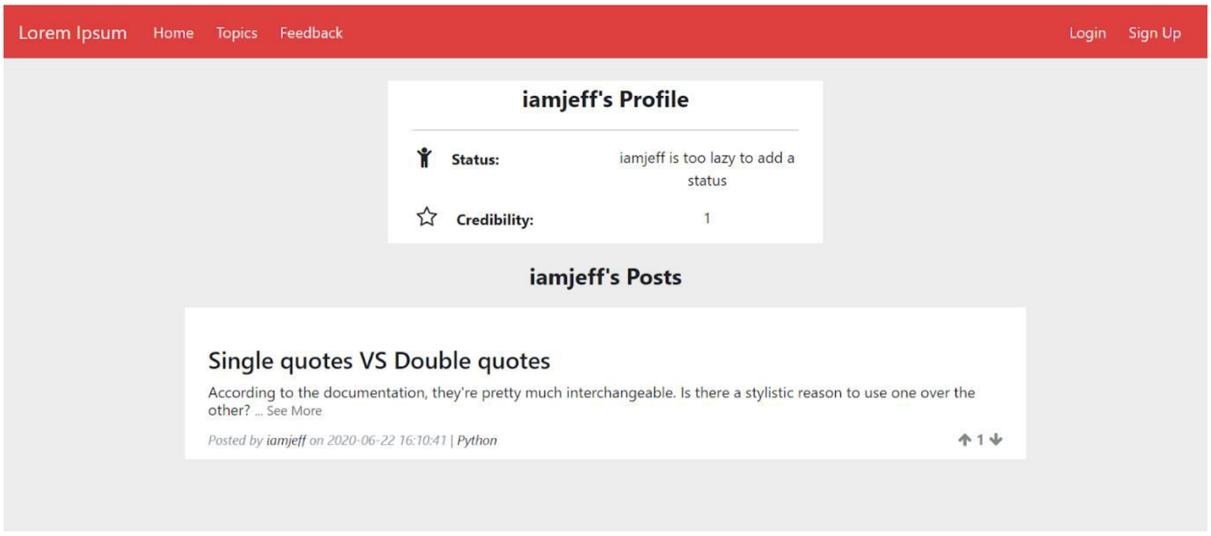
User has not posted anything

Each user already has a username to identify themselves on the website and it is not necessary for other users to know of a user's full name and birthday.

Solution 3: Display necessary data only

Display only the data necessary to identify a user on their profile such as their username.

Image 2.3.2 Profile page on secure website



iamjeff's Profile

Status:	iamjeff is too lazy to add a status
Credibility:	1

iamjeff's Posts

Single quotes VS Double quotes

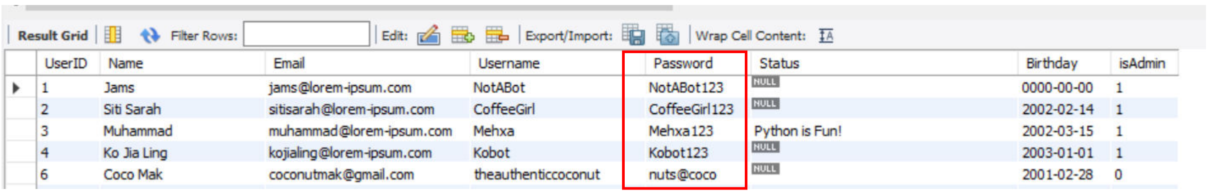
According to the documentation, they're pretty much interchangeable. Is there a stylistic reason to use one over the other? ... See More

Posted by iamjeff on 2020-06-22 16:10:41 | Python 1

Implementation 4: Passwords in database are not hashed/encrypted

If an attacker was able to gain access to a database or data that is not hashed/encrypted via Injection or other means, the accounts of users will be compromised as the login credentials of the users will exposed to the attackers.

Image 2.4.1 Plaintext credentials in database (vulnerable version)



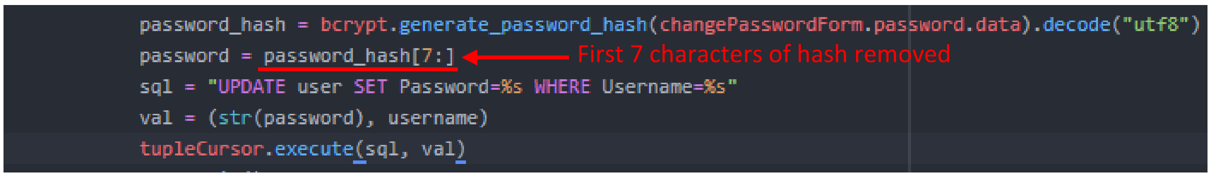
UserID	Name	Email	Username	Password	Status	Birthday	isAdmin
1	Jams	jams@lorem-ipsum.com	NotABot	NotABot123	NULL	0000-00-00	1
2	Siti Sarah	sitisarah@lorem-ipsum.com	CoffeeGirl	CoffeeGirl123	NULL	2002-02-14	1
3	Muhammad	muhammad@lorem-ipsum.com	Mehxa	Mehxa123	Python is Fun!	2002-03-15	1
4	Ko Jia Ling	kojialing@lorem-ipsum.com	Kobot	Kobot123	NULL	2003-01-01	1
6	Coco Mak	coconutmak@gmail.com	theauthenticcoconut	nuts@coco	NULL	2001-02-28	0

Solution 4: Hash passwords with strong hashing algorithms

A good hashing algorithm is irreversible. Hence, when passwords are hashed and password hashes are stored in the database instead of the plaintext passwords, the attacker will not be able to reverse the hashing function to get the password.

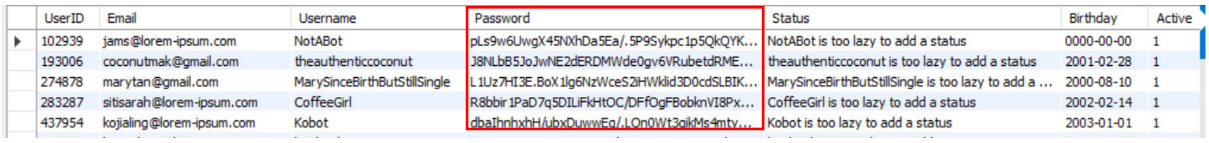
For this solution, I used a module called flask-bcrypt. Bcrypt is a recommended hashing function as it is specifically designed to be slow in order to stall attackers trying to brute force every combination of passwords to derive the hashed password. Moreover, Bcrypt automatically salts passwords, so no two hashed values would be the same. This will mislead attackers trying to find patterns in the different hash values. When a user signs up for the first time, I hash the user's password and store the hashed value into the database. Then when the user logs in to their account, the password that they input would be hashed and checked against the stored password hash in the database. This can easily done using Bcrypt's generate_password_hash() and check_password_hash() methods.

Image 2.4.2 Snippet of source code where password hash is generated and stored (secure version)



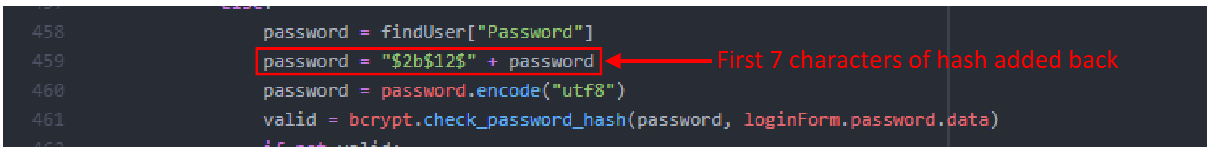
```
password_hash = bcrypt.generate_password_hash(changePasswordForm.password.data).decode("utf8")
password = password_hash[7:]
sql = "UPDATE user SET Password=%s WHERE Username=%s"
val = (str(password), username)
tupleCursor.execute(sql, val)
```

Image 2.4.3 Password hashes in database (secure version)



UserID	Email	Username	Password	Status	Birthday	Active
102939	jams@lorem-ipsum.com	NotABot	pLs9w6UwgX45NXhDa5Ea/.5P9Sykpc1p5QkQYK...	NotABot is too lazy to add a status	0000-00-00	1
193006	coconutmak@gmail.com	theauthenticcoconut	J8NLbB5JoJwNE2dERDMWde0gv6VRubetdRME...	theauthenticcoconut is too lazy to add a status	2001-02-28	1
274878	marytan@gmail.com	MarySinceBirthButStillSingle	L1Uz7HI3E.BoX1lg6NzWceS2iHWklid3D0cdSLBIK...	MarySinceBirthButStillSingle is too lazy to add a ...	2000-08-10	1
283287	sitisarah@lorem-ipsum.com	CoffeeGirl	R8bbir1PaD7q5DILiFkHtOC/DFfOgFBobknVI8Px...	CoffeeGirl is too lazy to add a status	2002-02-14	1
437954	kojialing@lorem-ipsum.com	Kobot	dbaIhnhxhH/ubxDuwwEg/.LOn0Wt3gikMs4mty...	Kobot is too lazy to add a status	2003-01-01	1

Image 2.4.4 Snippet of source code where input is verified with the stored password hash (secure version)



```
password = findUser["Password"]
password = "$2b$12$" + password
password = password.encode("utf8")
valid = bcrypt.check_password_hash(password, loginForm.password.data)
```

In this solution, I also removed the first 7 characters of a password hash before storing it into the database. This is due to the fact that all Bcrypt hashes start with “$2b$12$” and I did not want the attacker to recognise the hashing algorithm used as it may be easier for them to derive the password. However, I added the 7 characters back to the hash whenever a password needed to be verified.

Other Vulnerabilities

Apart from the Top 10 OWASP Vulnerabilities that I have chosen, I have also implemented other general security measures and practices to further improve the overall security of the application.

Solution 1: Improved Profile Update

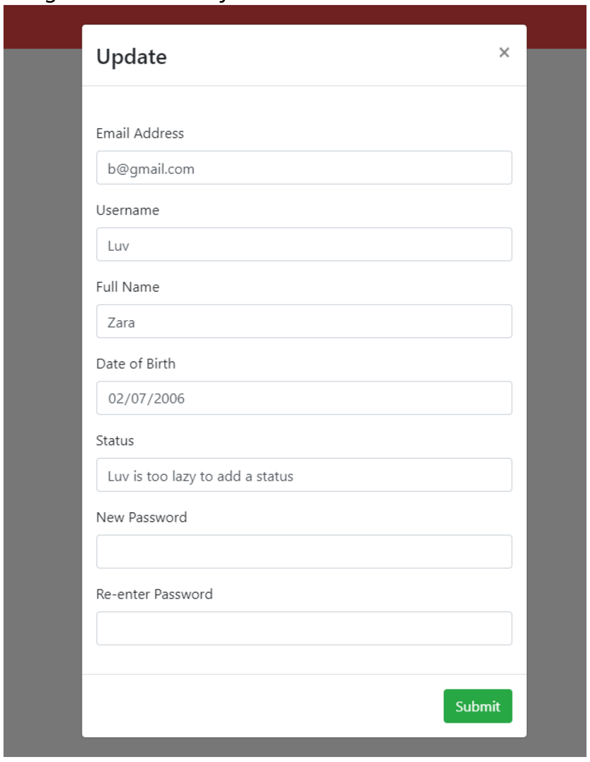
The profile update has been improved in two main ways: Update notifications and Password Changing Procedure.

In the vulnerable version of the website, the edit profile modal will display all profile information at once, exposing all sensitive data except password. In the secure version, I changed the layout such that you can only see and change one data field at one time. Moreover, I added an email feature so that when a user updates any of their information, they would receive an email notifying them that their account information has been changed.

This will help reduce the rate at which the attacker compromises personal data and notifies users of any unauthorized changes.

Additionally, the notification email contains a link which the user can use to change their password immediately if they feel that their account has been compromised.

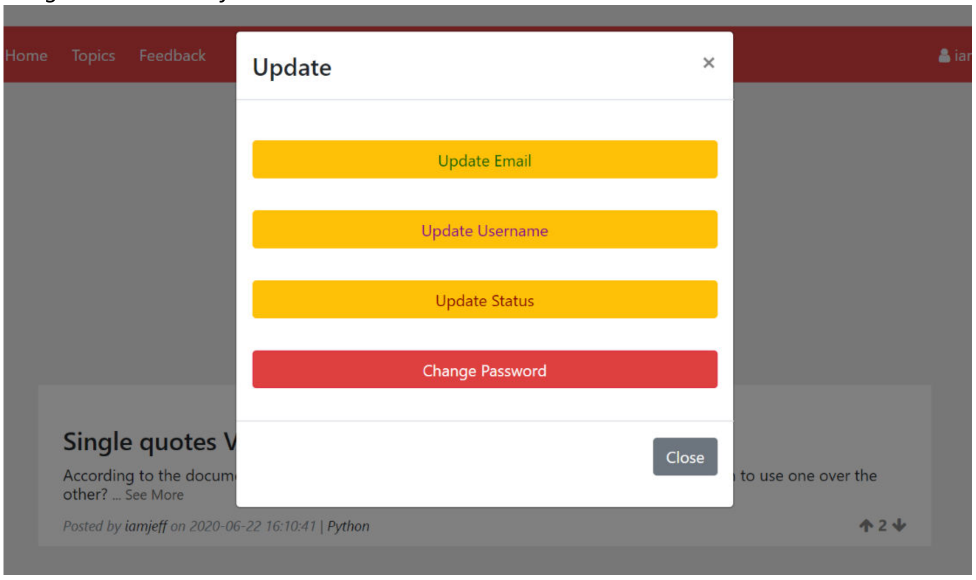
Image 3.1.1 Edit Profile Modal in vulnerable version



Update

Email Address	b@gmail.com
Username	Luv
Full Name	Zara
Date of Birth	02/07/2006
Status	Luv is too lazy to add a status
New Password	
Re-enter Password	

Image 3.1.2 Edit Profile modal in secure version

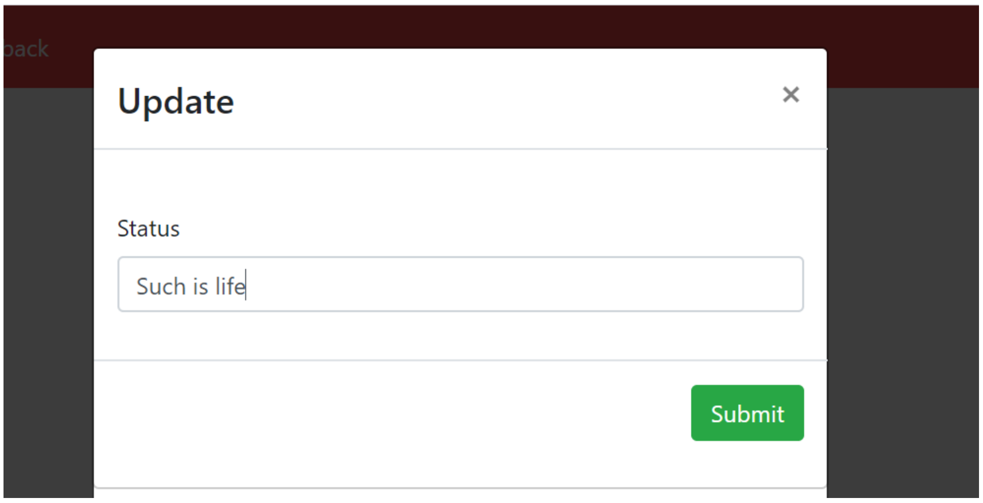


Single quotes V

According to the docum... to use one over the other? ... See More

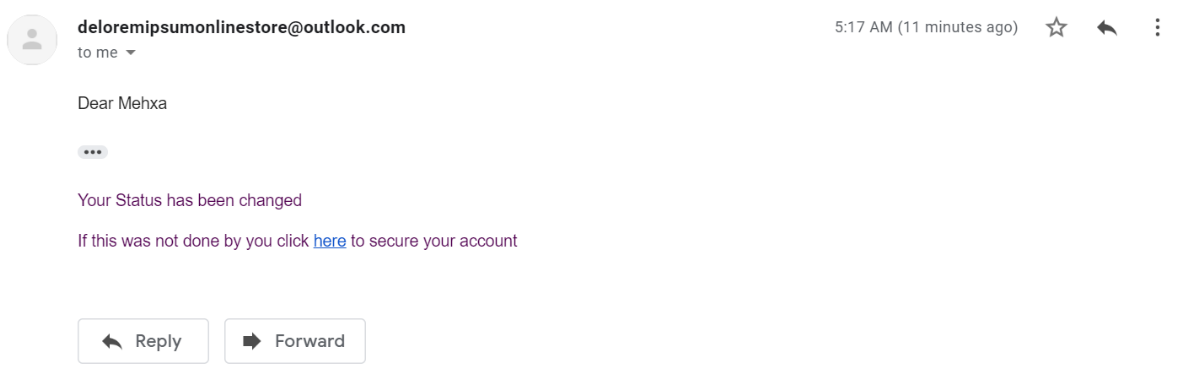
Posted by iamjeff on 2020-06-22 16:10:41 | Python

Image 3.1.3 User performing a status update



Such is life

Image 3.1.4 Email notification of status update



deloremipsumonlinestore@outlook.com

to me

5:17 AM (11 minutes ago)

Dear Mehxa

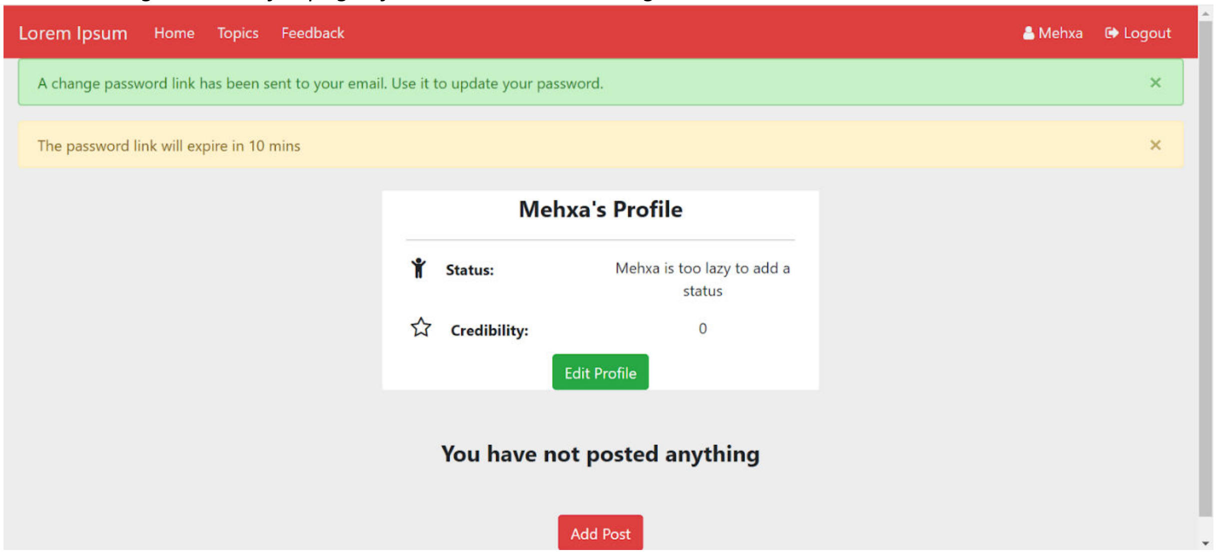
Your Status has been changed

If this was not done by you click here to secure your account

In the vulnerable version of the website, users will be able to change their passwords in the Edit Profile modal.

In the secure version, clicking on the change password button would send a reset password link to the user's email account. The link would only be valid for 10 minutes. The user must click on the link in their email to be sent to a temporary webpage where they can change their password. Clicking on the link after 10 minutes would redirect them back to the homepage with a flash message on the website telling them that their password reset link has expired.

Image 3.1.5 Profile page after user clicks on "Change Password" button



A change password link has been sent to your email. Use it to update your password.

The password link will expire in 10 mins

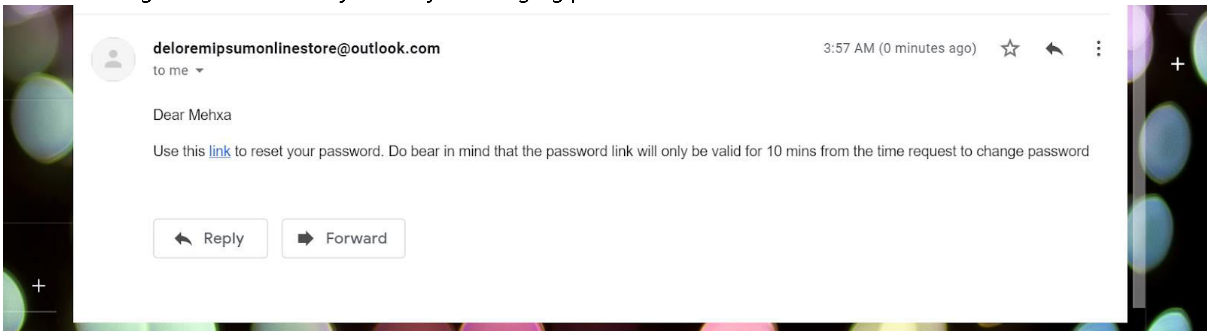
Mehxa's Profile

Status: Mehxa is too lazy to add a status

Credibility: 0

You have not posted anything

Image 3.1.6 Email notification for changing password

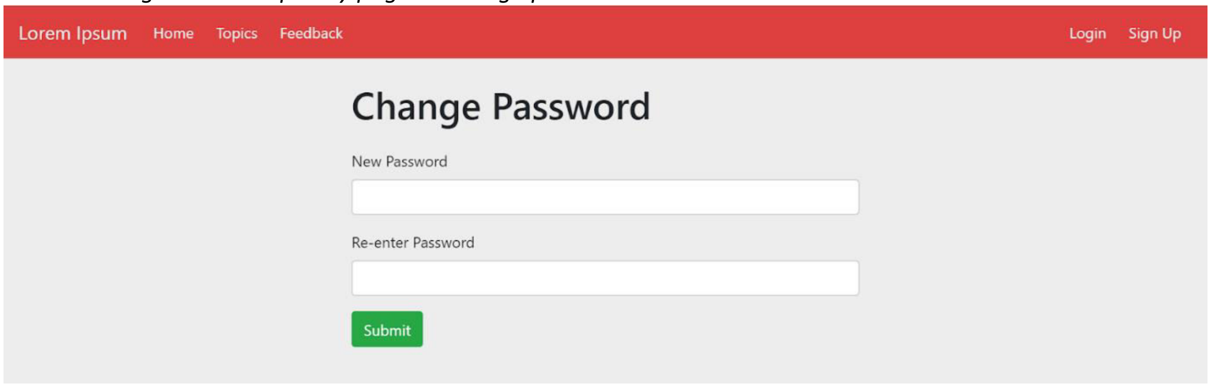


deloremipsumonlinestore@outlook.com
to me
3:57 AM (0 minutes ago)

Dear Mehxa

Use this link to reset your password. Do bear in mind that the password link will only be valid for 10 mins from the time request to change password

Image 3.1.7 Temporary page to change password

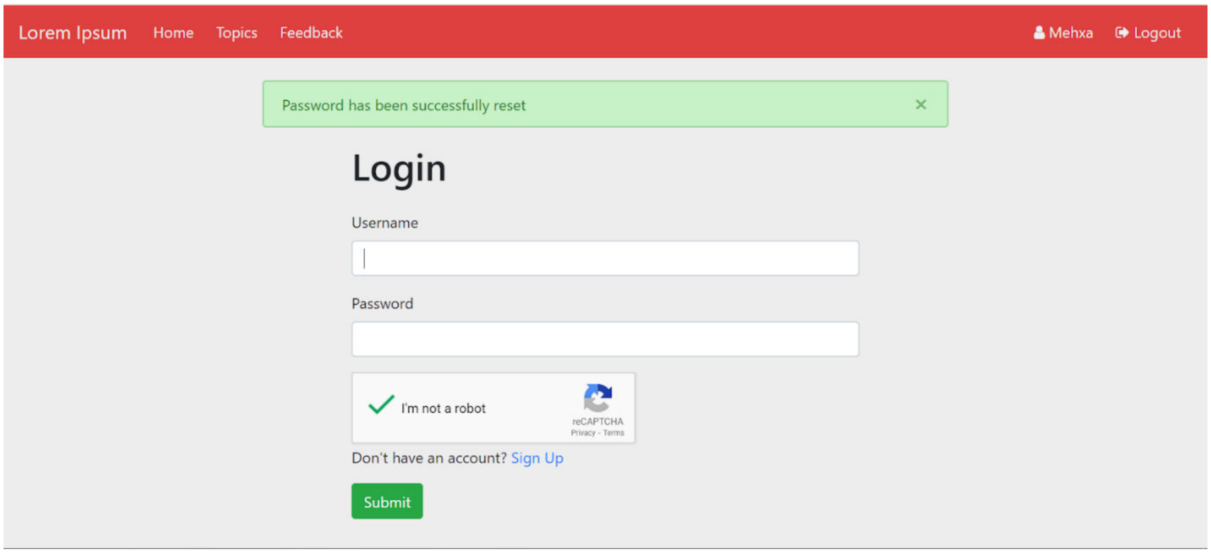


Change Password

New Password

Re-enter Password

Image 3.1.8 Successful Password Change



Lorem Ipsum Home Topics Feedback Mehxa Logout

Password has been successfully reset ×

Login

Username

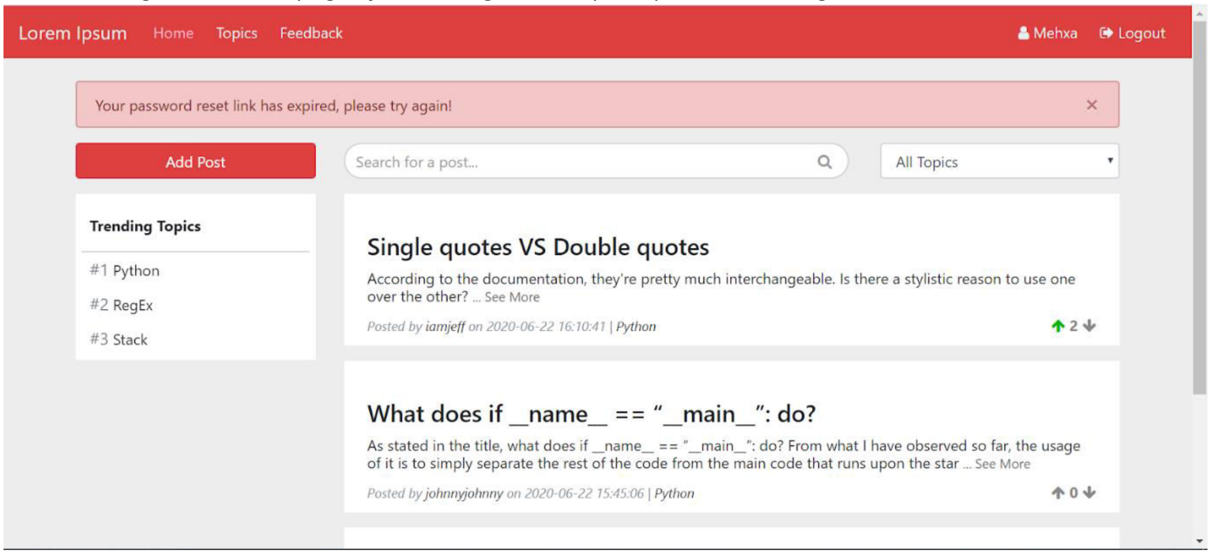
Password

I'm not a robot

Don't have an account? Sign Up

Submit

Image 3.1.9 Homepage after clicking on an expired password change link



Lorem Ipsum Home Topics Feedback Mehxa Logout

Your password reset link has expired, please try again! ×

Add Post

Search for a post...

All Topics

Trending Topics

#1 Python

#2 RegEx

#3 Stack

Single quotes VS Double quotes

According to the documentation, they're pretty much interchangeable. Is there a stylistic reason to use one over the other? ... See More

Posted by iamjeff on 2020-06-22 16:10:41 | Python 2

What does if __name__ == "__main__": do?

As stated in the title, what does if __name__ == "__main__": do? From what I have observed so far, the usage of it is to simply separate the rest of the code from the main code that runs upon the star ... See More

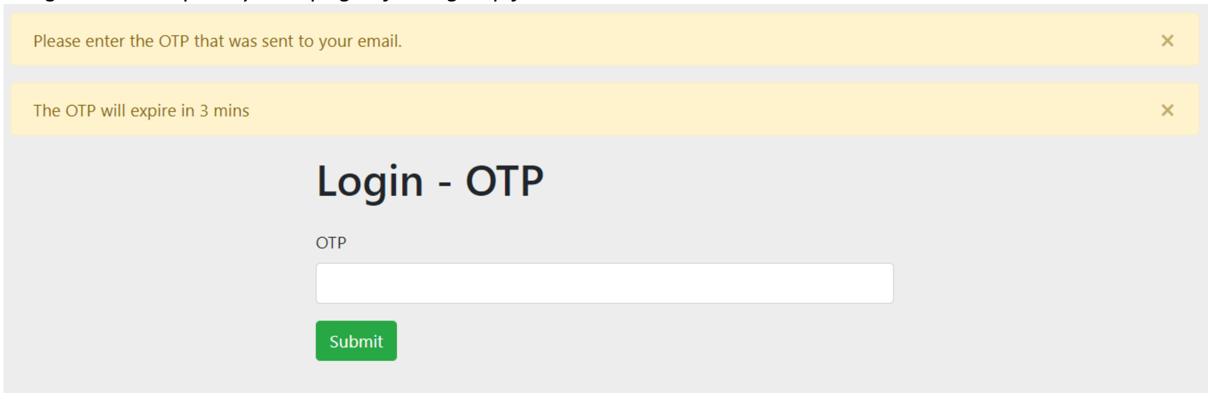
Posted by johnnyjohnny on 2020-06-22 15:45:06 | Python 0

Solution 2: OTP for Sign Up

In the secure version of the website, I added an OTP(One Time Password) feature to verify the user's email address. This is important as notifications and change password links will be sent to the user's email.

After the user fills up the sign up form and submits it, they will be redirected to a temporary website. An email will be sent to the email address that they have submitted and they would have to type in the OTP sent in the email. The OTP would only be valid for 3 minutes. Submitting the wrong OTP would trigger an error message and cause another OTP to be resent. Failing the OTP process thrice or entering an OTP 3 minutes after it has been sent would redirect the user to the sign up page and they would have to resubmit their sign up information.

Image 3.2.1 Temporary OTP page after sign up form has been submitted


Please enter the OTP that was sent to your email. ×

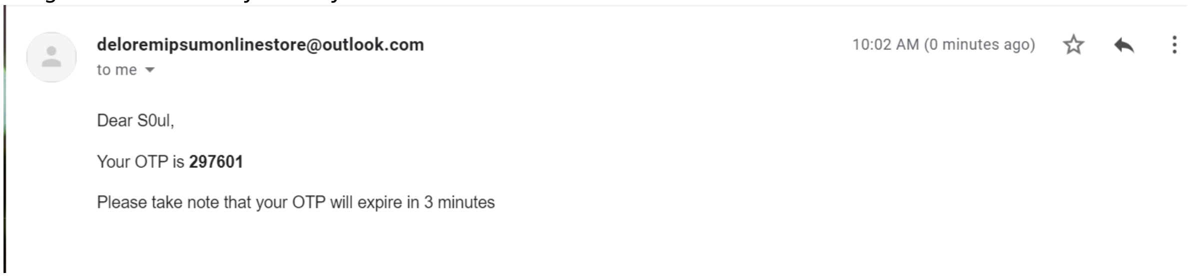
The OTP will expire in 3 mins ×

Login - OTP

OTP

Submit

Image 3.2.2 Email notification for OTP


deloremipsumonlinestore@outlook.com 10:02 AM (0 minutes ago)

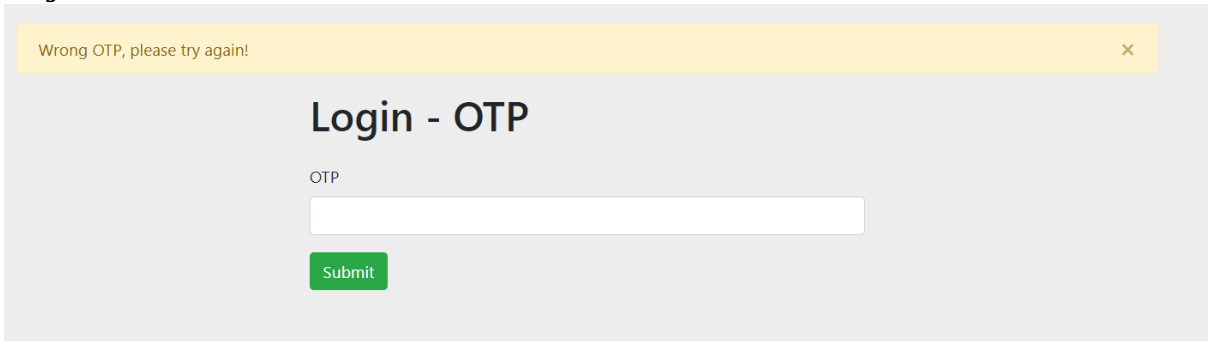
to me

Dear S0ul,

Your OTP is **297601**

Please take note that your OTP will expire in 3 minutes

Image 3.2.3 Incorrect OTP entered

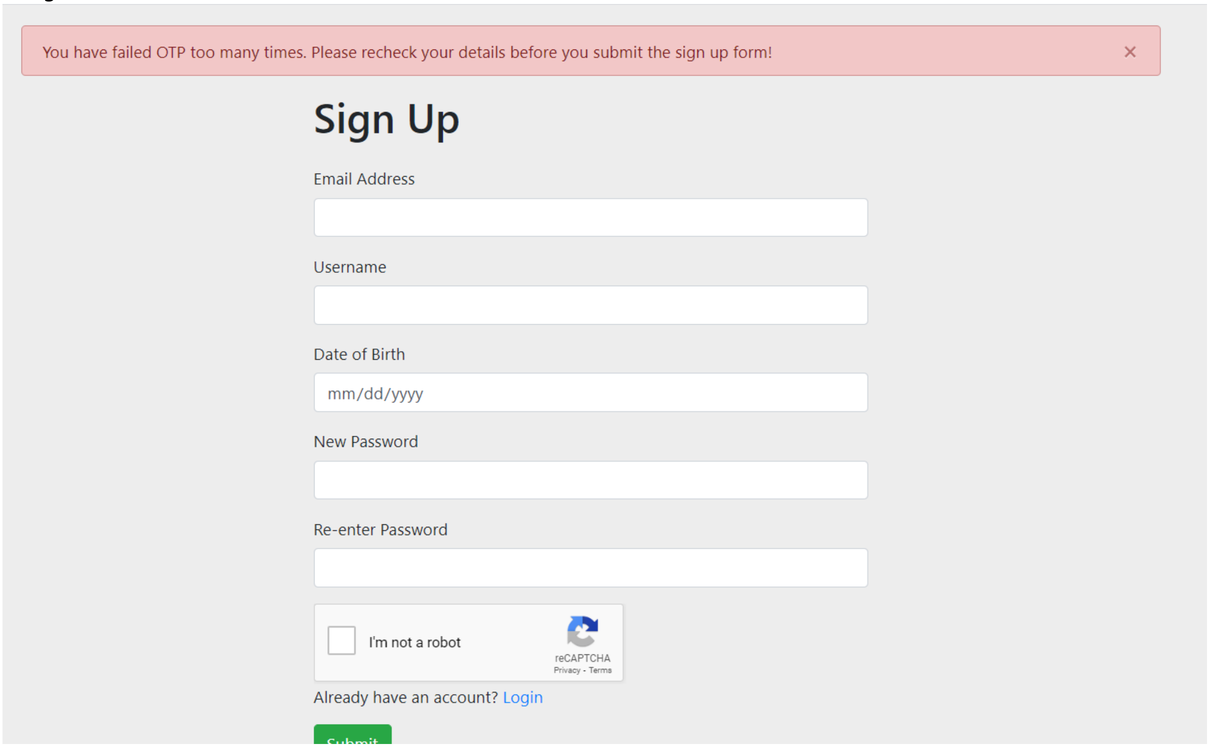

Wrong OTP, please try again! ×

Login - OTP

OTP

Submit

Image 3.2.4 Incorrect OTP entered 3 times


You have failed OTP too many times. Please recheck your details before you submit the sign up form! ×

Sign Up

Email Address

Username

Date of Birth

mm/dd/yyyy

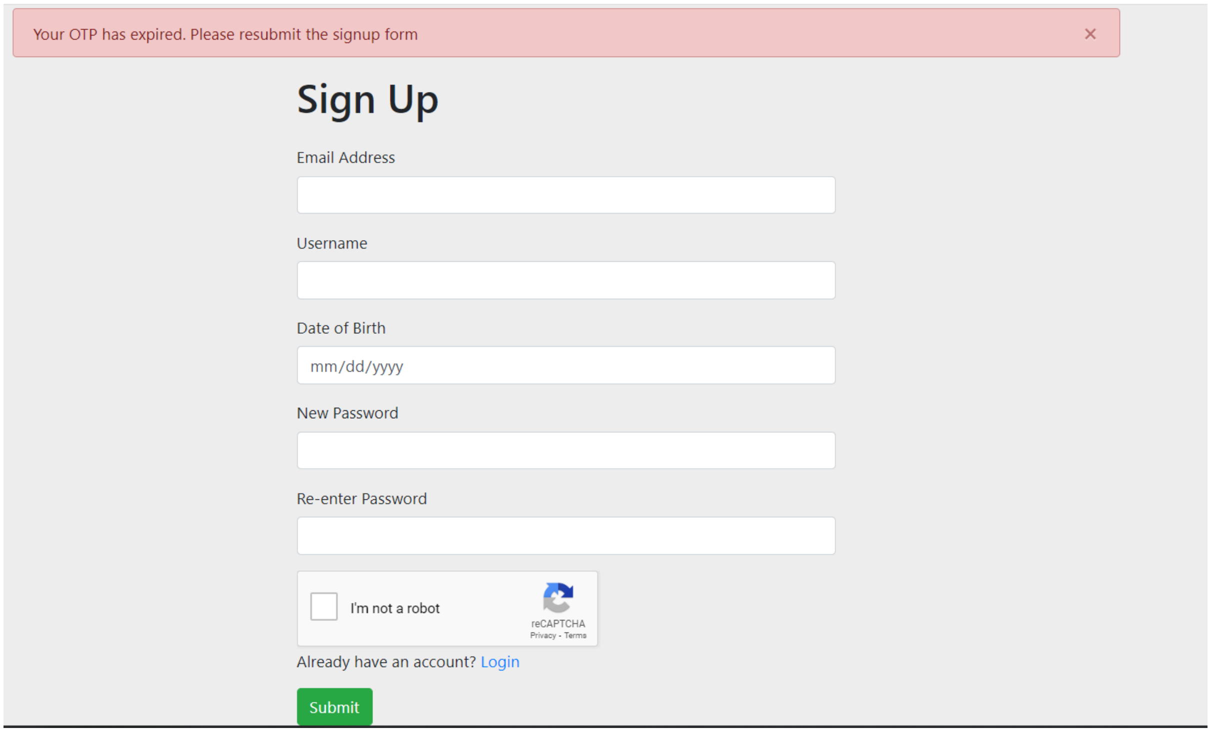
New Password

Re-enter Password

I'm not a robot

Already have an account? Login

Image 3.2.5 Entering OTP 3 minutes after it has been sent


Your OTP has expired. Please resubmit the signup form ×

Sign Up

Email Address

Username

Date of Birth

mm/dd/yyyy

New Password

Re-enter Password

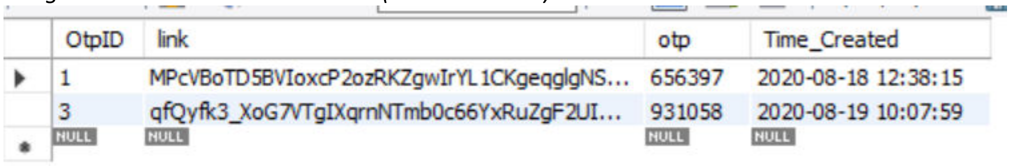
I'm not a robot

Already have an account? Login

Submit

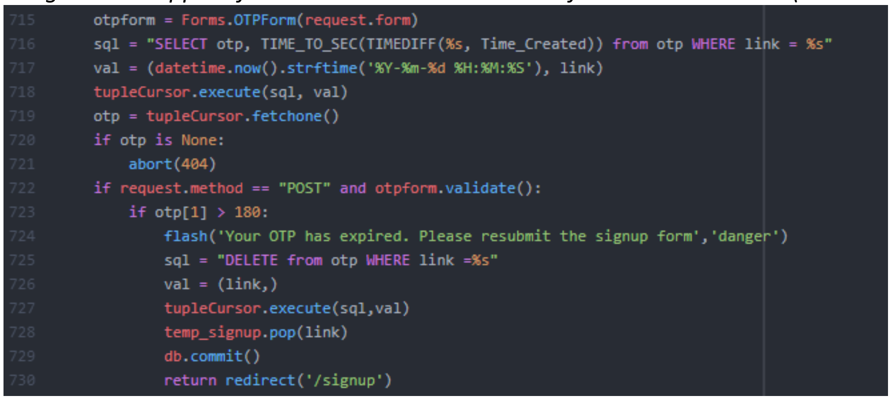
All temporary website URLs were created using the Python secrets module, using the method secrets.token_urlsafe(). The temporary URLs would be stored in the database, together with the time they were created. Once the user has used the URL or the time of the URL has expired, the URL would be deleted from the database. For the OTP table, when a new OTP is resent, the same link will be used but the value in the otp column will be updated to the new otp.

Image 3.2.6 OTP table in database (secure version)



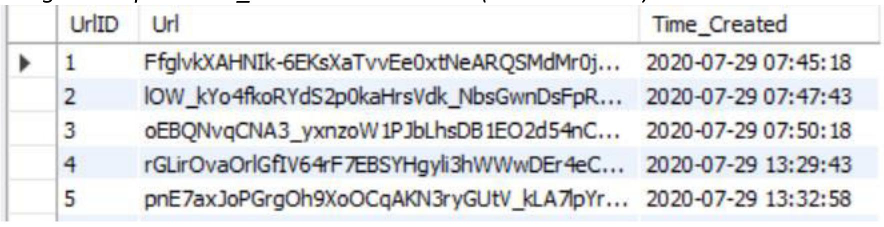
	OtpID	link	otp	Time_Created
▶	1	MPcVBoTD5BVIoxcP2ozRKZgwIrYL1CKgeqglgNS...	656397	2020-08-18 12:38:15
	3	qfQyfk3_XoG7VTgIXqrnNTmb0c66YxRuZgF2UI...	931058	2020-08-19 10:07:59
*	NULL	NULL	NULL	NULL

Image 3.2.7 Snippet of source code where the time of the OTP is checked (secure version)



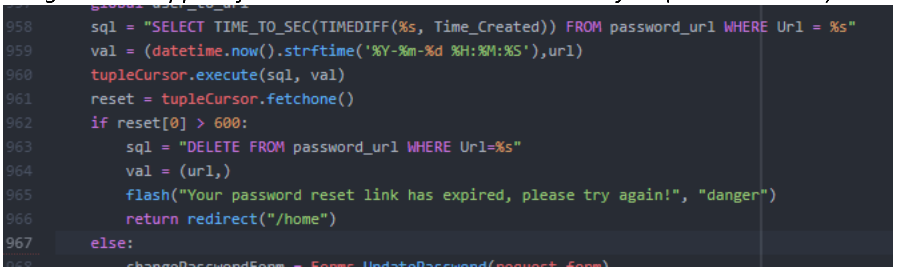
```
    otpform = Forms.OTPForm(request.form)
    sql = "SELECT otp, TIME_TO_SEC(TIMEDIFF(%s, Time_Created)) from otp WHERE link = %s"
    val = (datetime.now().strftime('%Y-%m-%d %H:%M:%S'), link)
    tupleCursor.execute(sql, val)
    otp = tupleCursor.fetchone()
    if otp is None:
        abort(404)
    if request.method == "POST" and otpform.validate():
        if otp[1] > 180:
            flash('Your OTP has expired. Please resubmit the signup form','danger')
            sql = "DELETE from otp WHERE link =%s"
            val = (link,)
            tupleCursor.execute(sql,val)
            temp_signup.pop(link)
            db.commit()
            return redirect('/signup')
```

Image 3.2.8 password_url table in database (secure version)



	UrlID	Url	Time_Created
▶	1	FfglvkXAHNIk-6EKsXaTvvEe0xtNeARQSMdMr0j...	2020-07-29 07:45:18
	2	lOW_kYo4fkoRYdS2p0kaHrsVdk_NbsGwnDsFpR...	2020-07-29 07:47:43
	3	oEBQNvqCNA3_yxnzoW1PJbLhsDB1EO2d54nC...	2020-07-29 07:50:18
	4	rGLirOvaOrlGfIV64rF7EBSYHgyli3hWWwDEr4eC...	2020-07-29 13:29:43
	5	pnE7axJoPGrgOh9XoOCqAKN3ryGUtV_kLA7lpYr...	2020-07-29 13:32:58

Image 3.2.9 Snippet of source code where the time is verified (secure version)



```
    sql = "SELECT TIME_TO_SEC(TIMEDIFF(%s, Time_Created)) FROM password_url WHERE Url = %s"
    val = (datetime.now().strftime('%Y-%m-%d %H:%M:%S'),url)
    tupleCursor.execute(sql, val)
    reset = tupleCursor.fetchone()
    if reset[0] > 600:
        sql = "DELETE FROM password_url WHERE Url=%s"
        val = (url,)
        flash("Your password reset link has expired, please try again!", "danger")
        return redirect("/home")
    else:
```

3. Other Individual Contributions

When we were first told about this assignment, I thought that it would be a mammoth task to complete and that it would be extremely difficult as all this while we've just been developing websites without paying much attention to the security aspect of development. Now, not only do we have to create 2 versions of the same website, but also secure one of them from the Top 10 OWASP Vulnerabilities.

I believe that our team has improved a lot in terms of efficiency and coding skills. We wanted to use an online template or reuse one of our old projects in order to save time, but I suggested that we should start from scratch as the template code may be done in languages that we may not know so we may not be able to mitigate the vulnerabilities well and our previous project used the Python shelve module as a database, which would not be ideal for Jialing to implement her Injection vulnerability and we may spend even more time rewriting our code just to make it fit the current project. After we agreed on it, my teammates worked extremely fast to create the base features and functionality of the project.

Due to their speed, I could only develop the base features for Profile page for both the user and admin accounts. Later in the project, I supplemented more on the other features such as login and sign up when implementing the additional security features.

I contributed more in integration and analysis as I explored the different features and packages of our dynamic scanning tool ZAP, to see if we can use it to its full potential and identify as many vulnerabilities as possible. I collated the dynamic scan reports from all members to create the overall scan report with all the vulnerabilities that the tool had identified.

I also contributed greatly in transferring the main functions from the vulnerable version of the application to the secure version of the application.

One of the biggest challenges I faced in this project would be when implementing encryption as a solution for Sensitive Data Exposure. The easiest way to accomplish this would be to implement HTTPS. However, there were many limitations. It is almost impossible to get a free digital certificate signed by a trusted authority. I found one website which gives free digital certificates and is a trusted authority. However, the website only supports flask servers that have Nginx or Apache, which would allow the website to be hosted on the internet. Since our website is locally hosted, the Certificate Authority was not able to issue us a free digital certificate. It is possible to generate a self-signed certificate. However, just like in the implementation above, it would still not be considered secure as it was not signed by a Trusted Certificate Authority. Hence in order to replicate HTTPS to the best of my ability, I resorted to using a strong encryption algorithm so that the data bring transferred can still be encrypted.

There were many considerations that I had to make when choosing an appropriate encryption algorithm:

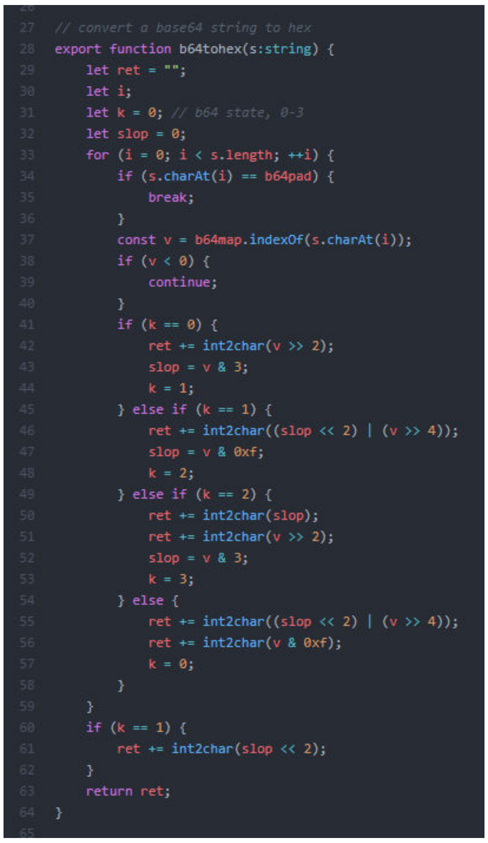
- Security - Whether the algorithm is secure enough and meets the recommended standards
- Feasibility - Whether the algorithm can be implemented into our application without causing any problems
- Technical aspect - Whether the algorithm requires extra frameworks or language support that we may not have or were not choosing to implement

- Interoperability – Since we are using Python Flask as the server and the client - side encryption would have to use JavaScript, the chosen algorithm must be able to work well on both languages.
- Time – The encryption algorithm used must not take too much time to carry out the encryption process, otherwise the application would run extremely slowly.

I spent most of my time doing research and experimenting with the many frameworks and codes available online only to be very disappointed as most of the frameworks required Node.js, which none of us knew how to use. Moreover, there were many libraries in Python for different types of PKI encryption such as RSA and ECC (Elliptic Curve Cryptography) but those were Python – only implementations and was not compatible to work with JavaScript.

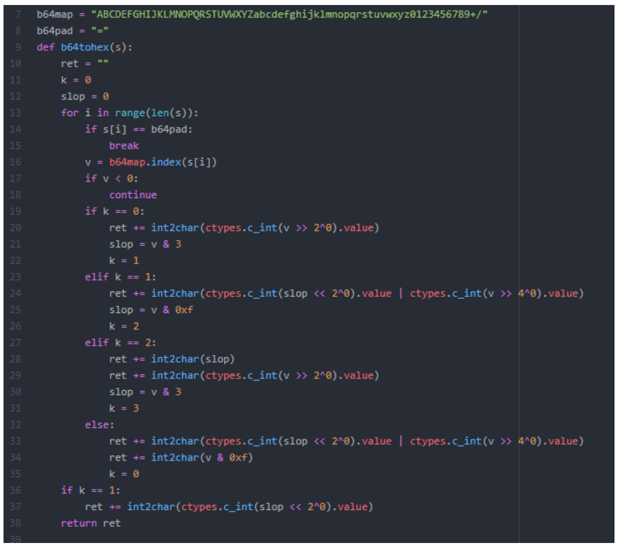
Initially I wanted to use ECC as it provided the same amount of security as RSA but with a smaller key size. A 256 bit ECC key encryption is equivalent to a 3072 bit RSA key encryption process. However, as I mentioned earlier, there were many Python libraries for ECC and all the JavaScript versions required Node.js. There were also not many projects that used Python Flask and JavaScript for encryption (I could only find one such project and even that project used some aspects of Node.js). Eventually I settled with a JavaScript RSA library called JSEncrypt. However, since it was a full JavaScript library, I had to convert some of the essentials features of JSEncrypt in Python so that it can work on both languages. Images 4.1 and 4.2 show an example of a function that I had to convert from JavaScript to Python.

Image 4.1 JSEncrypt code



```
// convert a base64 string to hex
export function b64tohex(s:string) {
    let ret = "";
    let i;
    let k = 0; // b64 state, 0-3
    let slop = 0;
    for (i = 0; i < s.length; ++i) {
        if (s.charAt(i) == b64pad) {
            break;
        }
        const v = b64map.indexOf(s.charAt(i));
        if (v < 0) {
            continue;
        }
        if (k == 0) {
            ret += int2char(v >> 2);
            slop = v & 3;
            k = 1;
        } else if (k == 1) {
            ret += int2char((slop << 2) | (v >> 4));
            slop = v & 0xf;
            k = 2;
        } else if (k == 2) {
            ret += int2char(slop);
            ret += int2char(v >> 2);
            slop = v & 3;
            k = 3;
        } else {
            ret += int2char((slop << 2) | (v >> 4));
            ret += int2char(v & 0xf);
            k = 0;
        }
    }
    if (k == 1) {
        ret += int2char(slop << 2);
    }
    return ret;
}
```

Image 4.2 My implementation in Python



```python
b64map = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
b64pad = "="
def b64tohex(s):
    ret = ""
    k = 0
    slop = 0
    for i in range(len(s)):
        if s[i] == b64pad:
            break
        v = b64map.index(s[i])
        if v < 0:
            continue
        if k == 0:
            ret += int2char(ctypes.c_int(v >> 2^0).value)
            slop = v & 3
            k = 1
        elif k == 1:
            ret += int2char(ctypes.c_int(slop << 2^0).value | ctypes.c_int(v >> 4^0).value)
            slop = v & 0xf
            k = 2
        elif k == 2:
            ret += int2char(slop)
            ret += int2char(ctypes.c_int(v >> 2^0).value)
            slop = v & 3
            k = 3
        else:
            ret += int2char(ctypes.c_int(slop << 2^0).value | ctypes.c_int(v >> 4^0).value)
            ret += int2char(v & 0xf)
            k = 0
    if k == 1:
        ret += int2char(ctypes.c_int(slop << 2^0).value)
    return ret
```

Overall, I feel that this project has opened my eyes to the security aspect of development which I did not previously have any experience with and I am very proud of our team for being able to accomplish this mammoth task within the deadline while also having to deal with other major assignments in other modules. Personally, I would like to take the upcoming semester break to learn new frameworks and languages such as Node.js so that I would not be limited by my lack of skills in future projects.

- End of Document -