COMP6443 Web Application Security

UNSWTalk Security Assessment

Something Awesome

Leo Liu – z5080336
5-1-2018

Contents

Vulnerability Classification

P1 – CRITICAL

Vulnerabilities that cause a privilege escalation on the platform from unprivileged to admin, allows remote code execution, financial theft, etc. Examples: vulnerabilities that result in Remote Code Execution such as Vertical Authentication bypass, SSRF, XXE, SQL Injection, User authentication bypass.

P2 – HIGH

Vulnerabilities that affect the security of the platform including the processes it supports. Examples: Lateral authentication bypass, Stored XSS, some CSRF depending on impact.

P3 – MEDIUM

Vulnerabilities that affect multiple users, and require little or no user interaction to trigger. Examples: Reflective XSS, Direct object reference, URL Redirect, some CSRF depending on impact.

P4 – LOW

Issues that affect singular users and require interaction or significant prerequisites (MitM) to trigger. Examples: Common flaws, Debug information, Mixed Content.

P5 – ACCEPTED RISK

Non-exploitable weaknesses and “won’t fix” vulnerabilities. Examples: Best practices, mitigations, issues that are by design or acceptable business risk to the customer such as use of CAPTCHAS.

Vulnerabilities

vID: 1

Date Discovered: 12th May 2018

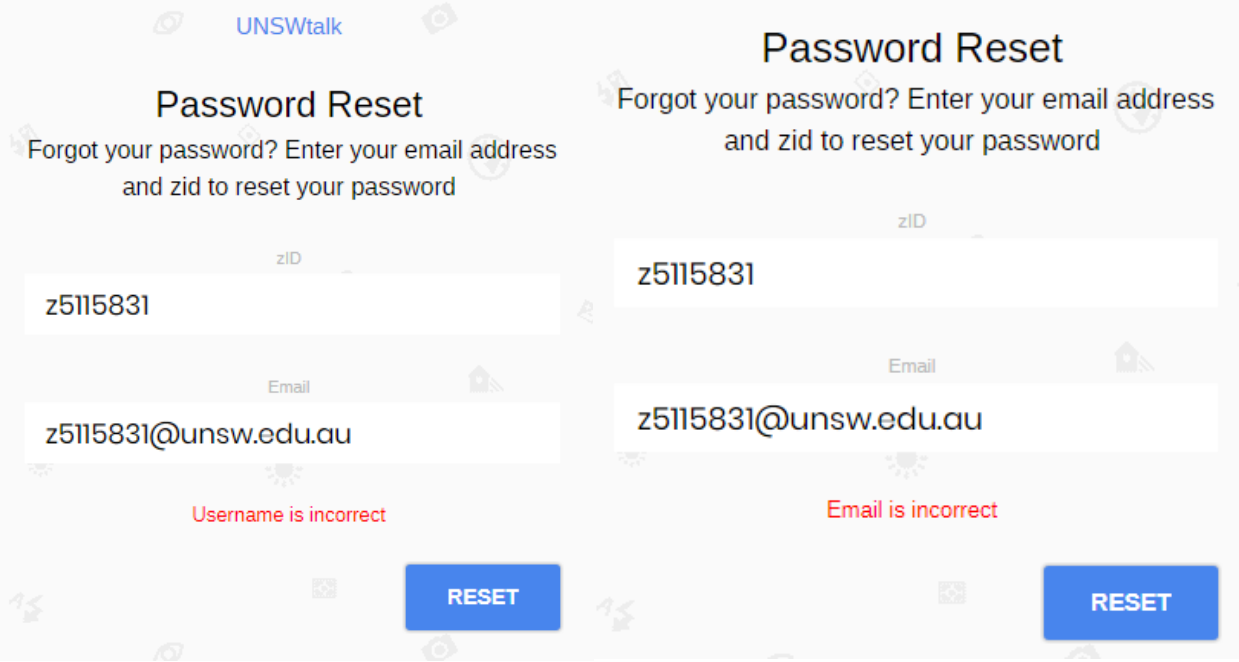
URL: /forgetPassword

Severity: P4

Type: Brute Force Information Disclosure

Result: Discover a user's username and email

Description: The system informs you if you entered in a valid username, and then if you entered in a valid email. There is no coded delay for verifying the username and email so a brute force attack could be carried out very quickly.



vID: 2

Date Discovered: 16th May 2018

URL: /createAccount

Severity: P4

Type: Brute Force Information Disclosure

Result: Discover a user's username and email

Description: The system informs you if you entered in a valid username or valid email. There is no coded delay for verifying the username and email so a brute force attack could be carried out very quickly.

---

vID: 3

Date Discovered: 13th May 2018

URL: /*

Severity: P2

Type: Security Misconfiguration

Result: Discover information of all users

Description: Directory listing is on. The files and directories in the root directory is visible if you simply visit the root directory in your browser, which allows anyone to see all the structure of the project, as well as all user's usernames, and amount of posts they made

Index of /~z5080336/2041/UNSWtalk

Name	Last modified	Size	Description
Parent Directory	16-May-2018 12:40	-	
UNSWtalk.cgi	16-Nov-2017 18:35	1k	
UNSWtalk.py	12-May-2018 21:17	25k	
UNSWtalkDatabase.db	16-May-2018 00:18	80k	
__pycache__/	12-May-2018 21:18	-	
dataset-medium/	22-Feb-2018 15:53	-	
dataset-small/	22-Feb-2018 15:53	-	
diary.txt	16-Nov-2017 18:35	2k	
files.tar	16-Nov-2017 18:35	1.4M	
functions.py	15-Nov-2017 16:32	20k	
init	16-Nov-2017 18:35	1k	
initDB.py	16-Nov-2017 18:35	6k	
initFiles.py	16-Nov-2017 18:35	1k	
log.txt	16-Nov-2017 18:35	19k	
old_profile_page.txt	16-Nov-2017 18:35	10k	
requirements.txt	16-Nov-2017 18:35	1k	
setTaggedPosts.py	16-Nov-2017 18:35	3k	
static/	22-Feb-2018 15:53	-	
strdup.h	16-Nov-2017 18:35	1k	
structure.py	16-Nov-2017 18:35	6k	
submit	16-Nov-2017 18:35	1k	
templates/	22-Feb-2018 15:53		

---

vID: 4

Date Discovered: 14th May 2018

URL: /profile

Severity: P3

Type: Denial of Service

Result: Prevents data to be written to disk

Description: Uploading files for cover photo and profile image has no size limit. These files are stored on the server so an attacker could upload large files to prevent other users from using the website or uploading their own images.

---

vID: 5

Date Discovered: 15th May 2018

URL: /*

Severity: P4

Type: Debug Information

Result: Reveals server-side code

Description: The user can trigger server-side errors such as inserting “a” as the pageNum parameter when viewing paginated pages. When they do, a detailed report of errors will be displayed to the user, including snippets of server-side code. If the code is secure, this should not result in any significant exploits (the code is good 😊)

builtins.ValueError

```
ValueError: invalid literal for int() with base 10: 'nan'
```

Traceback (most recent call last)

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1997*, in __call__

```
return self.wsgi_app(environ, start_response)
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1985*, in wsgi_app

```
response = self.handle_exception(e)
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1540*, in handle_exception

```
reraise(exc_type, exc_value, tb)
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/_compat.py", line *33*, in reraise

```
raise value
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1982*, in wsgi_app

```
response = self.full_dispatch_request()
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1614*, in full_dispatch_request

```
rv = self.handle_user_exception(e)
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1517*, in handle_user_exception

```
reraise(exc_type, exc_value, tb)
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/_compat.py", line *33*, in reraise

```
raise value
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1612*, in full_dispatch_request

```
rv = self.dispatch_request()
```

File "/tmp_amd/adams/export/adams/1/cs2041/public_html/Python-3.6.3/lib/python3.6/site-packages/flask/app.py", line *1598*, in dispatch_request

```
return self.view_functions[rule.endpoint](**req.view_args)
```

File "/tmp_amd/reed/export/reed/3/z5080336/public_html/2041/UNSWtalk/functions.py", line *427*, in wrapper

```
return func(*args, **kwargs)
```

File "/tmp_amd/reed/export/reed/3/z5080336/public_html/2041/UNSWtalk/UNSWtalk.py", line *218*, in home

```
pageNum = int(pageNum)
```

ValueError: invalid literal for int() with base 10: 'nan'

The debugger caught an exception in your WSGI application. You can now look at the traceback which led to the error.

To switch between the interactive traceback and the plaintext one, you can click on the "Traceback" headline. From the text traceback you can also create a paste of it.

vID: 6

Date Discovered: 15th May 2018

URL: /*

Severity: P3

Type: Insecure HTTP

Result: Capture user data

Description: Username, password, and all other user info and other data can be sent from client to server (and vice versa) using port 80 HTTP instead of port 443 Secure HTTP. Firefox, being a good Samaritan, pops out numerous warnings when you try to login using http mode.

---

vID: 7

Date Discovered: 16th May 2018

URL: /*

Severity: P2

Type: XSS

Result: CSRF, Phishing, DoS

Description: All content written by the user as a post are evaluated as HTML

```
home.html
54        {% for result in postResults %}
55            <script>
56                $("#posts").append(messageBlock('{{result.fromUser}}', '{{result.id}}', '{{result.fromUser}}', '
                      {{result.time}}', `{{result.message}}`, '{{result.children|length}}', '{{result.taggedStr}}', '
                      {{result.likes|length}}'))
```

---

vID: 8

Date Discovered: 16th May 2018

URL: /*

Severity: P2

Type: XSS

Result: CSRF, Phishing, DoS

Description: All content written by the user as a comment are evaluated as HTML

```
home.html
54        {% for result in postResults %}
55            <script>
56                $("#posts").append(messageBlock('{{result.fromUser}}', '{{result.id}}', '{{result.fromUser}}', '
                      {{result.time}}', `{{result.message}}`, '{{result.children|length}}', '{{result.taggedStr}}', '
                      {{result.likes|length}}'))
```

---

vID: 9

Date Discovered: 16th May 2018

URL: /*

Severity: P2

Type: XSS

Result: CSRF, Phishing, DoS

Description: Name of users displayed are evaluated as HTML. Allows users to create usernames with a HTML string to perform XSS attacks.

```
home.html
54        {% for result in postResults %}
55            <script>
56                $("#posts").append(messageBlock('{{result.fromUser}}', '{{result.id}}', '{{result.fromUser}}', '
                      {{result.time}}', `{{result.message}}`, '{{result.children|length}}', '{{result.taggedStr}}', '
                      {{result.likes|length}}'))
```

---

vID: 10

Date Discovered: 21st May 2018

URL: /verifyAccount

Severity: P4

Type: Brute Force Information Disclosure

Result: Discover a user's username and verify code

Description: The system informs you if you supplied a correct username or not when you attempt to enter the verify link, if the account exists, it informs you the verify code does not match. This function does not have a time delay so attackers can quickly find a list of all users by viewing the server response from z0000000 to z9999999.

```python
@app.route('/verifyAccount', methods=['GET', 'POST'])
def verifyAccount():
    verify_code = request.args.get('verify_code')
    zid = request.args.get('zid')
    user = DBSession.query(User).filter_by(zid=zid).first()
    if not user:
        return "Account for " + zid + " not found"
    if user.verify_code != verify_code:
        return "Verify code " + verify_code + " does not match " + user.verify_code
    user.account_verified = True
```

---

vID: 11

Date Discovered: 21st May 2018

URL: /verifyAccount

Severity: **P1**

Type: RCE

Result: REMOTE CODE EXECUTION omg

Description: When a user is verified, the system will create a directory for them to store their uploaded images, with the following code:

```python
os.system("mkdir " + os.path.join("static", "dataset", user.zid))
```

user.zid is a user supplied username for themself, upon testing with

```
>>> path = os.path.join("static", "dataset", "hi; mkdir hello_world")
>>> os.system("mkdir " + path)
mkdir: cannot create directory ‘static/dataset/hi’: No such file or directory
0
>>> exit()
fried:/mnt/c/Users/fried/Desktop/Something_Awesome/test_verify_code_os $ ls
hello_world
```

It is evident that supplying a username such as “; mkdir hello_world” will create a directory called hello world. The user can then supply dangerous usernames such as “; rm -rf /”

---

vID: 12

Date Discovered: 22nd May 2018

URL: /api

Severity: P2

Type: Lateral Authentication Bypass

Result: Freely obtain other user's private details

Description: A target user's details as well as their posts can be displayed to any other logged in user, even if the target user set their privacy setting to hide all posts and details from other users (or limit all details and posts to friends only) by visiting /api?data=userDetails&action=get&zid=<user>

```
UNSWtalk.py
        elif(data == "userDetails"):
            if(action == "get"):
                if(values == "json"):
                    if(zid == "all"):
                        #print([user.toDict() for user in DBSession.query(User).all()])
                        return json.dumps([user.toDict() for user in DBSession.query(User).all()])
                    else:
                        return json.dumps(DBSession.query(User).filter_by(zid=zid).first().toDict())
                else:
                    if(zid == "all"):
                        return DBSession.query(User).all()
                    else:
                        return DBSession.query(User).filter_by(zid=zid).first()
```

---

vID: 13

Date Discovered: 22nd May 2018

URL: /*

Severity: P2

Type: Plain Text Passwords

Result: Passwords St0L3n

Description: Since the server has so many vulnerabilities listed already, storing the passwords in plain text is extremely dangerous as it could easily get leaked.

```
student.txt
2 password: amanda
```

---

vID: 14

Date Discovered: 22nd May 2018

URL: /api

Severity: P2

Type: Lateral Authentication Bypass

Result: Freely delete any other user's posts

Description: A target user's posts and comments can be removed by any other user by simply visiting the api page like /api?data=posts&action=remove&zid=<any number>

```
UNSWtalk.py
            elif(action == "remove"):
                postID = request.args.get("zid")
                DBSession.query(Post).filter_by(id=postID).delete()
                return "success"
```

---

vID: 15

Date Discovered: 22nd May 2018

URL: /api

Severity: P2

Type: Lateral Authentication Bypass

Result: Freely do anything as another user

Description: the api page is so full of vulnerabilities as it doesn't check authentication at all and allows you to do everything as another user by simply supplying a different zid when using the api page

```
UNSWtalk.py
@app.route('/api/', methods=['GET', 'POST'])
def api():
    if request.method == "GET":
        zid = request.args.get("zid")
        data = request.args.get("data")
        action = request.args.get("action")
        values = request.args.get("values")
```

Tests Completed

tID: 1

Date Tested: 10th March 2018

URL: /login

Test: SQL Injection

Aim: To determine if SQL injection is possible for the username and password fields

Method: Testing various SQL injection strings for the username & password fields, as well as reviewing the underlying code

Result: SQL injection at /login username & password is through sqlalchemy queries. Sqlalchemy is pretty good.No SQL injection possible.

functions.py

```
# Checks if the username matches the password
def checkLogin(zid, password):
    user = DBSession.query(User).filter_by(zid=zid).first()
    if not user:
        return False
    if(user.password == password and user.account_verified == True):
        return True
    else:
        return False
```

---

tID: 2

Date Tested: 11th May 2018

URL: /forgotPassword

Test: RCE

Aim: To determine if the send_main() function in /forgotPassword is vulnerable to RCE since the user can enter variables for the 'mutt' system call

Method: Copied function to a smaller program and tested various data as the 'to' and 'message' field to try to get a remote code execution

Result: RCE not possible as subprocess run only runs the first program and the rest is passed through as string arguments.

```python
functions.py
def send_email(to, subject, message):

    mutt = [
            'mutt',
            '-s',
            subject,
            '-e', 'set copy=no',
            '-e', 'set realname=UNSWtalk',
            '-e', 'set content_type=text/html',
            '--', to
    ]

    subprocess.run(
            mutt,
            input = message.encode('utf8'),
            stderr = subprocess.PIPE,
            stdout = subprocess.PIPE,
    )
```

---

tID: 3

Date Tested: 12th May 2018

URL: /api

Test: Stored XSS

Aim: To determine if the image names users upload are able to act as a stored XSS when the image is loaded into the page

Method: Uploaded various files with different file names that could potentially be dangerous as well as investigating the underlying code

Result: The test revealed the name of the file the user is uploading does not matter as all files are renamed to 'profileImage.jpg' or 'profileImage.png'

---

tID: 4

Date Tested: 20th May 2018

URL: /createAccount

Test: SQL Injection

Aim: To determine if there is a set of strings the user can enter to trigger a SQL injection when creating a new user

Method: Testing various SQL injection strings for the username, email and password fields, as well as reviewing the underlying code.

Result: SQL injection is not possible since SQLalchemy deals with it.

```
UNSWtalk.py
            newUser = User(
                zid=zid,
                email=email,
                full_name=full_name,
                password=password,
                verify_code=verify_code,
                account_verified=False
            )
            DBSession.add(newUser)
            DBSession.commit()
```

---

tID: 5

Date Tested: 21st May 2018

URL: /verifyAccount

Test: Authentication Bypass

Aim: To determine if a user is able to bypass verifying their email address, since the code looks kinda dodgy

Method: Test supplying no username, no verify codes to see if it passes the if statement.

Result: If SQLAlchemy does not find a user, it will most certainly execute line 191 and return before any more of the code is executed. Line 192 will throw an error if the user object is 'blank' and doesn't contain a verify_code. There is no way to generate a user object with a blank verify_code attribute to match with a supplied blank verify_code.

```
UNSWtalk.py
185 @app.route('/verifyAccount', methods=['GET', 'POST'])
186 def verifyAccount():
187     verify_code = request.args.get('verify_code')
188     zid = request.args.get('zid')
189     user = DBSession.query(User).filter_by(zid=zid).first()
190     if not user:
191         return "Account for " + zid + " not found"
192     if user.verify_code != verify_code:
193         return "Verify code " + verify_code + " does not match " + user.verify_code
194     user.account_verified = True
```

Bugs Found

bID: 1

Date Found: 14th May 2018

URL: /profile

Description: Users are able to upload files which are not of an image format, resulting in the image not being displayed, but instead an error icon being displayed.

---

bID: 2

Date Found: 15th May 2018

URL: /, /search

Description: Home and search uses pagination, suppling a string or blank value into the ?page= query will result in the server crashing (TypeError 'null' or 'string' is not an int)

UNSWtalk.py

```
223        pageNum = int(pageNum)
```

---

bID: 3

Date Found: 16th May 2018

URL: /*

Description: When hovering over username at top right corner to obtain dropdown menu, the mouse will clip the loading strip under the header causing the dropdown menu to disappear.

---

bID: 4

Date Found: 16th May 2018

URL: /createAccount, /forgotPassword

Description: CSS align does not properly align the elements in the center where they should be, also the loading bar does not disappear even after page load.

Repairs Applied

bID: 1

Date Repaired: 23rd May 2018

Repairs Completed: Made the file upload input only accept image files, as well as trying to create a javascript image object with supplied image before storing permanently on disk and displaying image

profile.html

```
<input type = "file" name = "file" accept="image/*" onChange="validateAndUpload(this);" />
```

profile.html

```
{% block pre_scripts %}
<script>
    function validateAndUpload(input){
        var URL = window.URL || window.webkitURL;
        var file = input.files[0];

        if (file) {
            var image = new Image();

            image.onload = function() {
                if (this.width) {
                    console.log('Image has width, I think it is real image');
                    //TODO: upload to backend
                }
            };

            image.src = URL.createObjectURL(file);
        }
    };
</script>
{% endblock %}
```

bID: 2

Date Repaired: 24th May 2018

Repairs Completed: Added a try except block to return the home page if the pageNum throws an exception

```python
@app.route('/', methods=['GET','POST'])
@login_required
def home():
    pageNum = request.args.get("page")

    try:
        pageNum = int(pageNum)
    except:
        return redirect(url_for("home", page=1))
```

---

vID: 1

Date Repaired: 21st May 2018

Repairs Completed: On an incorrect zID or email, still appear as if it "succeeded" to the client, but not actually send the email. This will prevent brute-force attacks from checking whether the supplied username / email is valid.

---

vID: 2

Date Repaired: 21st May 2018

Repairs Completed: To make brute force checking details slower, each time you try to create an account, there will be a 5 second delay. This will only (hopefully) sleep the thread of the attacker so no other users will be affected. Alongside DDoS protection offered by CloudFlare, it should be fairly hard to exploit this vulnerability now.

```python
# Creates an account
@app.route('/createAccount', methods=['POST', 'GET'])
def createAccount():
    if request.method == "POST":
        time.sleep(5)
```

---

vID: 3

Date Repaired: 22nd May 2018

Repairs Completed: Added a .htaccess file with directory index off `Options -Indexes`. Now when you try to view the directory you get the error

Access Forbidden

This is an "Error Code 403" message.

You were looking for the file
/~z5080336/2041/UNSWtalk/
which you do not have permission to access.

Please check the URL of the document you are after and make sure it has been entered correctly.

If the error persists, please contact z5080336@cse.unsw.edu.au with a copy of this error message.

Enjoy the CSE website.

Error Information

Requested URL:	**/~z5080336/2041/UNSWtalk/**
Error Status:	**403**
Error notes:	**Directory index forbidden by rule: /web/z5080336/2041/UNSWtalk/**
Request Protocol:	**HTTP/1.1**
Server Name:	**cgi.cse.unsw.edu.au**
Server Port:	**443**

---

vID: 4

Date Repaired: 22nd May 2018

Repairs Completed: Javascript will validate file size to be under 2MB per image so users cannot upload large files.

```
profile.html
<input type = "file" name = "file" accept="image/*" onChange="validateAndUpload(this);" />
```

```
if (this.files[0].size > 2097152) {
    alert("File is too big")
}
```

---

vID: 5

Date Repaired: 23rd May 2018

Repairs Completed: Copied website into local directory to work on development versions. Live version will have debug turned off in flask settings.

```
if __name__ == '__main__':
    app.secret_key =
    atexit.register(closeDBSession)
    app.run(debug=False, host="127.0.0.1", port=9777)
```

---

vID: 6

Date Repaired: 23rd May 2018

Repairs Completed: Added HTTP to HTTPS redirect in .htaccess

```
  GNU nano 2.2.6              File: .htaccess                          Modified

RewriteEngine On
# This will enable the Rewrite capabilities

RewriteCond %{HTTPS} !=on
# This checks to make sure the connection is not already HTTPS

RewriteRule ^/?(.*) https://%{HTTP_HOST}%{REQUEST_URI}/$1 [R,L]
# This rule will redirect users from their original location, to the same locat$
# i.e.  http://www.example.com/foo/ to https://www.example.com/foo/
# The leading slash is made optional so that this will work either in httpd.conf
# or .htaccess context

Options -Indexes
AddDefaultCharset utf-8




                              [ Read 16 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Secure | https://cgi.cse.unsw.edu.au/~z5080336/2041/UNSWtalk/UNSWtalk.cgi/login

---

vID: 7, 8, 9

Date Repaired: 23rd May 2018

Repairs Completed: Jinja2 escape messages using {{ msg | e }}

```
home.html
{% for result in postResults %}
    <script>
        $("#posts").append(messageBlock('{{result.fromUser|e}}', '{{result.id|e}}', '{{result.fromUser|e}}', '
            {{result.time}}', `{{result.message|e}}`, '{{result.children|length}}', '{{result.taggedStr|e}}', '
            {{result.likes|length}}'))
```

---

vID: 10

Date Repaired: 24th May 2018

Repairs Completed: Verify Account will redirect you to the home page no matter if it's verified or not. Legit users will have their account automatically verified while malicious users won't know if the account was verified or not.

```python
@app.route('/verifyAccount', methods=['GET', 'POST'])
def verifyAccount():
    verify_code = request.args.get('verify_code')
    zid = request.args.get('zid')
    user = DBSession.query(User).filter_by(zid=zid).first()
    if not user:
        return redirect(url_for("login"))
        #return "Account for " + zid + " not found"
    if user.verify_code != verify_code:
        return redirect(url_for("login"))
        #return "Verify code " + verify_code + " does not match " + user.verify_code
```

---

vID: 11

Date Repaired: 24th May 2018

Repairs Completed: instead of os.system(), subprocess.run() is used similar to the send emails test.

```python
    mkdir = [
        'mkdir',
        os.path.join("static", "dataset", user.zid)
    ]

    subprocess.run(mkdir)
```

---

vID: 12

Date Repaired: 25th May 2018

Repairs Completed: if requesting user is not target user, target's user privacy checked before sending any response for the api request.

```python
            if (session.get("current_user") != zid):
                if (DBSession.query(User).filter_by(zid=zid).first().settings.privacy != PrivacyLevel.PUBLIC):
                    return {}
```

---

vID: 12, 13, 14

Date Repaired: 25th May 2018

Repairs Completed: current user's identity checked with the target user (or owner of a post or account) before authorizing the action

```
412 ▾        elif(action == "remove"):
413              postID = zid
414 ▾            if (session.get("current_user") == DBSession.query(Post).filter_by(id=postID).first().owner):
415                  DBSession.query(Post).filter_by(id=postID).delete()
416                  return "success"
417              else:
418                  return "not authorized"
```