ERSS HW3 -- Security Report

Zi Xiong

Yunhe Wang

0. Abstract

Security is crucial for a server software. There are always malicious clients who try to attack the server software and many users will suffer from that once attacks succeed. Therefore, it is really important to develop a secure server software to prevent malicious attacks. In this report, we will analyze HW1 completed by zw154 and zq24, and HW2 completed by zg50 and cw373 from perspectives of both security and code quality.

1. Review/Analysis

HW1:

Quality

Abstraction:

In general, the developer used good abstraction for realizing clear source code. There are 3 different apps: “accounts” manages the information of users, “driverRegister” implements the function of becoming a driver and “rideRequest” handles ordering works. Additionally, there is a “template” directory along with app directories which is mainly used to display the navigation links for the whole website. This strongly improve the interface because the user is able to go to subjective pages at anywhere. They also reasonably specified URLs within appropriate apps so the whole code is organized. They have User, Driver and RequestRide models to perform data exchange. They didn’t directly use Django original views like DeleteView but self-written this kind of views, which make the code logically clearer.

Readability:

The code readability is pretty good: The name assigned to functions, variables and URLs were appropriate and meaningful. A suggestion is that they can sometimes delete commented-out code since they frequently comment out some parts of codes. Also, sometimes the code doesn’t follow PEP 8 rules and tiny errors were shown when browsing the code. Although this is a trivial problem, changing of it is really helpful for improving the overall code readability.

Documentation:

The team kept recording the danger log during the development of software, which is great. They identified security errors related to Login verification, Database deletion, Email sending. A suggestion is that they can also include some potential risks of suffering from security attacks such as MITM, XSS, DOS.

Understandability:

The code understandability is really good: There are no complicated external packages used or self-defined execution files to make the code hard to understand. The developer followed Django MVT patterns to reasonably combine front-end and back-end works.

Other:

- The share request part of the code use the same template as ordinary ride request part, which make the share request to be allowed “only if the share request perfectly match the ride request”, which is unreasonable. Therefore, the programmer should individually write these two parts and not to repeat the code here.
- Logical Fallacies: The ride request can specify past date and the driver can accept that request; The driver is able to accept his own request;
- Email functionality is not implemented.
- From user interface perspective, it is better to always show the username of current logged in user. Also, the format of input should be told to users otherwise it is too confusing.

Security

XSS (cross site scripting):

From https://code.djangoproject.com/wiki/AutoEscaping, “At the moment, {{ name }} outputs the following: <script>alert('XSS');</script>. With auto escaping, this will be output as: &lt;script&gt;alert(&apos;XSS&apos;);&lt;/script&gt;” Basically, Django has the functionality to convert the risky characters as “<” to other harmless strings to prevent attackers from injecting malicious scripts. It has disadvantages as if a user set the class as {{ 'class1 onmouseover=javascript:func()’ }} or using custom template tags like is_safe, the risk will significantly rise. Nevertheless, the programmer didn’t use this and so that the software is strongly protected against XSS.

CSRF (cross-site request forgeries) + MITM (man-in-the-middle):

From https://docs.djangoproject.com/en/2.2/ref/csrf/, “The CSRF middleware and template tag provides easy-to-use protection against Cross Site Request Forgeries”. The Django itself has a strong “csrf_tokens” to prevent CSRF attack. From analyzing source code of CSRF prevention code https://docs.djangoproject.com/en/2.2/_modules/django/middleware/csrf/, it is shown that even with MITM, the attacker cannot successfully attack the server. The programmer used {csrf_token} for every “post” request so that even a malicious user inject scripts for posting use, nothing will happen. In addition, Django force developer to have a way to protect against CSRF since version 1.2, so CSRF attack is considered very seriously and the attacker had almost no chance to do this attack.

DoS (Denial of Service):

There is no good way to fundamentally prevent DoS. The programmer didn't implemented file upload functionality, so it can avoid direct DoS. However, as user use specific tools to send huge amounts of request to the server, it will inevitably down. The effective way to protect is to shut down the dubious IP address which might DoS attacking.

SQL Injection:

From https://docs.djangoproject.com/en/2.2/topics/security/, "Django's querysets are protected from SQL injection since their queries are constructed using query parameterization. A query's SQL code is defined separately from the query's parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver." Therefore, as the programmer implement the operation of data into the database through Django ORM, the risk of suffering SQL injection will be highly reduced. In this app, all database operations are done by ORM so it is highly unlikely can be attacked by SQL injection. Moreover, the programmer developed user registration page with Django login form, which not allow users to input invalid characters. That is, even with SQL injection risks, the website can potentially prevent malicious and catastrophic operation.

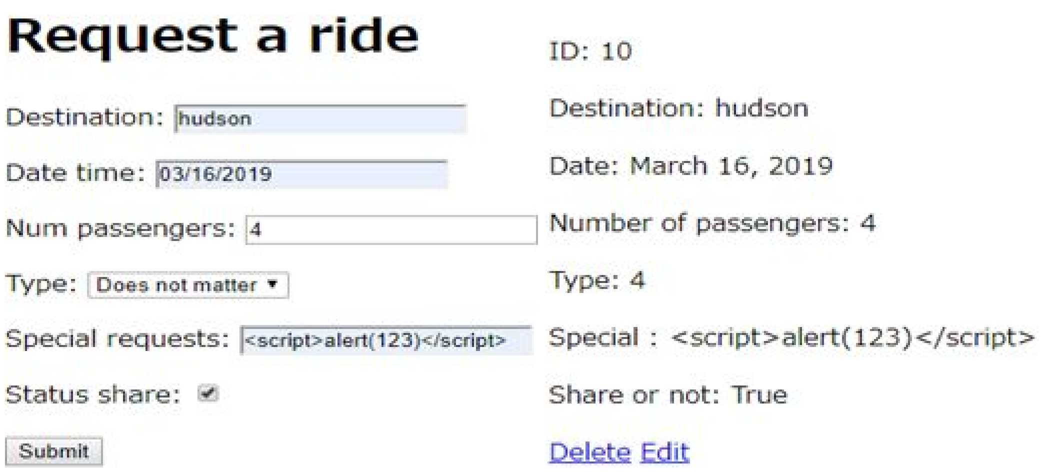
Login Authentication:

This website needs logged-in status to perform most of work as ride request, drive confirming, share ride, etc. Therefore, a case that users try to do these things while they are logged out is needed to be considered. Django has @login_required decorator to force user to login before accessing specific pages. The programmer also noticed this problem and stated in danger log, but he applied it just to the profile but not to other pages, and it is definitely wrong from the perspective of security.

By these analysis, the software should be vulnerable to Dos but immune to XSS, CSRF and SQL Injection.

2. Attacks

XSS:



Request a ride

Destination:	hudson
Date time:	03/16/2019
Num passengers:	4
Type:	Does not matter ▾
Special requests:	<script>alert(123)</script>
Status share:	✓

Submit

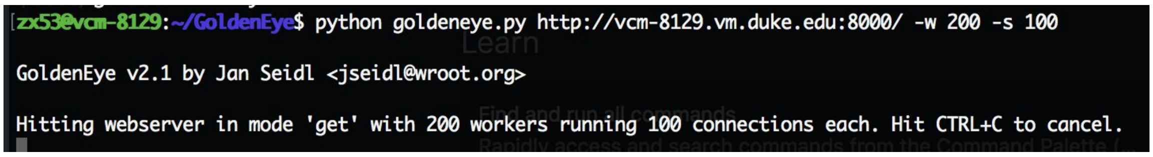
ID: 10
Destination: hudson
Date: March 16, 2019
Number of passengers: 4
Type: 4
Special : <script>alert(123)</script>
Share or not: True
Delete Edit

Figure 1: Test of XSS attack on request ride page

In figure 1, we tried to insert a script: <script>alert(123)</script> to try to make the pop up of “123”, but it is just converted to a plain text by Django auto escaping functionality.

DoS:

We took a DDoS attack test with the ride sharing website in HW1 on our own Duke virtual machine using a open source program called GoldenEye. As in figure 2, we set up 200 workers to visit the website concurrently, each worker with 100 concurrent sockets.



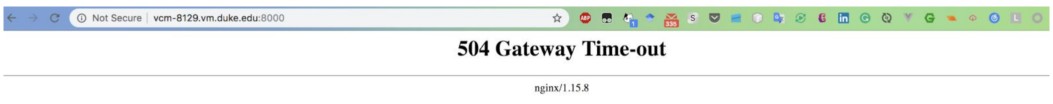
```
zx53@vcm-8129:~/GoldenEye$ python goldeneye.py http://vcm-8129.vm.duke.edu:8000/ -w 200 -s 100

GoldenEye v2.1 by Jan Seidl <jseidl@wroot.org>

Hitting webserver in mode 'get' with 200 workers running 100 connections each. Hit CTRL+C to cancel.
```

Figure 2: DDoS attack test using GoldenEye

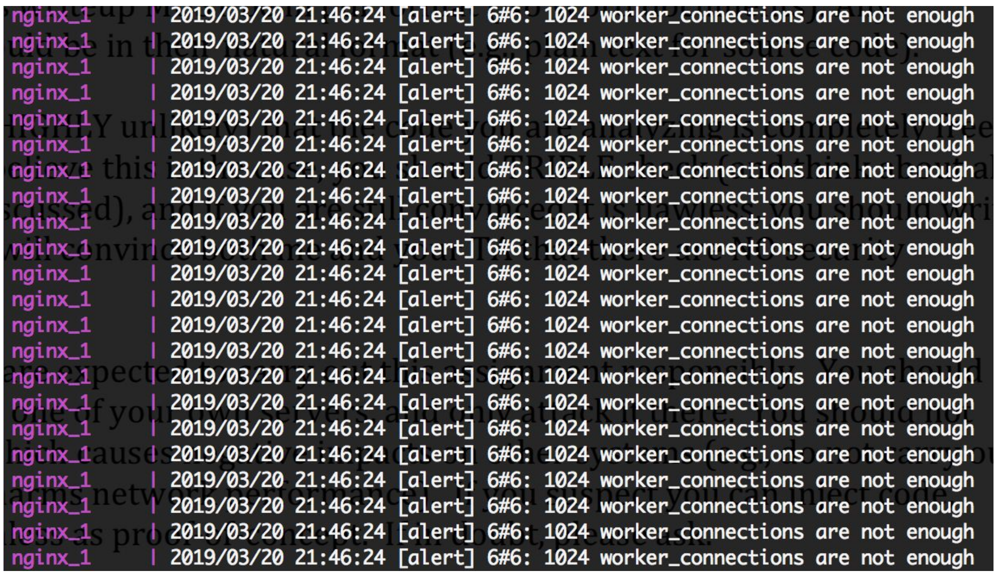
During the DDoS attack, as in the figure 3, we found that the server is no longer accessible (we received 504 Gateway Time-out responses instead of normal responses). As in figure 4, the server log also showed that nginx could not support so many concurrent connections.


Not Secure | vcm-8129.vm.duke.edu:8000

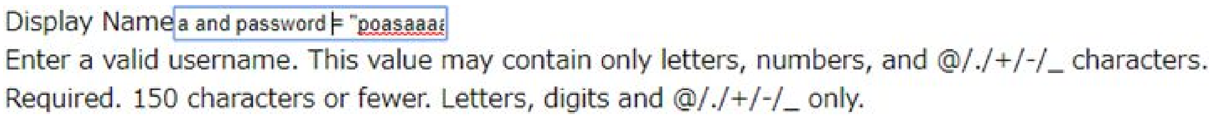
504 Gateway Time-out

nginx/1.15.8

Figure 3: 504 Gateway Time-out due to DDoS attack


```
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
nginx_1      | 2019/03/20 21:46:24 [alert] 6#6: 1024 worker_connections are not enough
```
Figure 4: nginx warnings due to DDoS attack

SQL Injection:


Display Name a and password = "poasaaaa

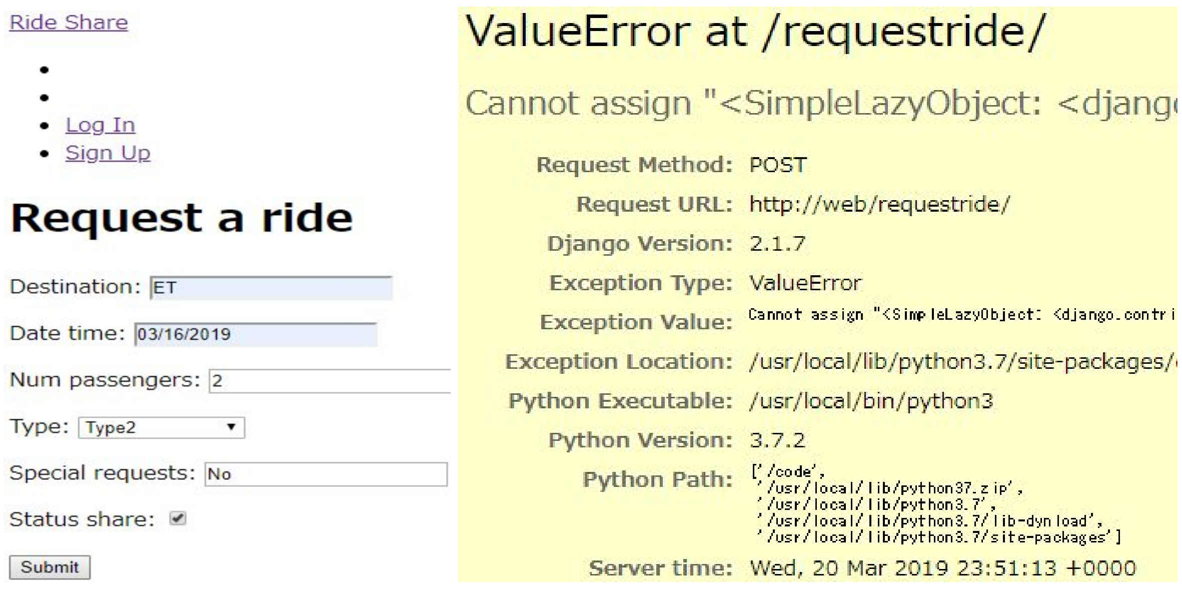
Enter a valid username. This value may contain only letters, numbers, and @/./+/-/_ characters.

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Figure 5: Test of SQL injection attack on register page

In Figure 5, I tried to set the password in Display Name section by SQL injection, but it is not successful since “=” character is not allowed here, that is it is hard to construct SQL relationship in WHERE part.

Login Authentication:


Ride Share

-
-
- Log In
- Sign Up

Request a ride

Destination: ET

Date time: 03/16/2019

Num passengers: 2

Type: Type2

Special requests: No

Status share: ✓

Submit

ValueError at /requestride/

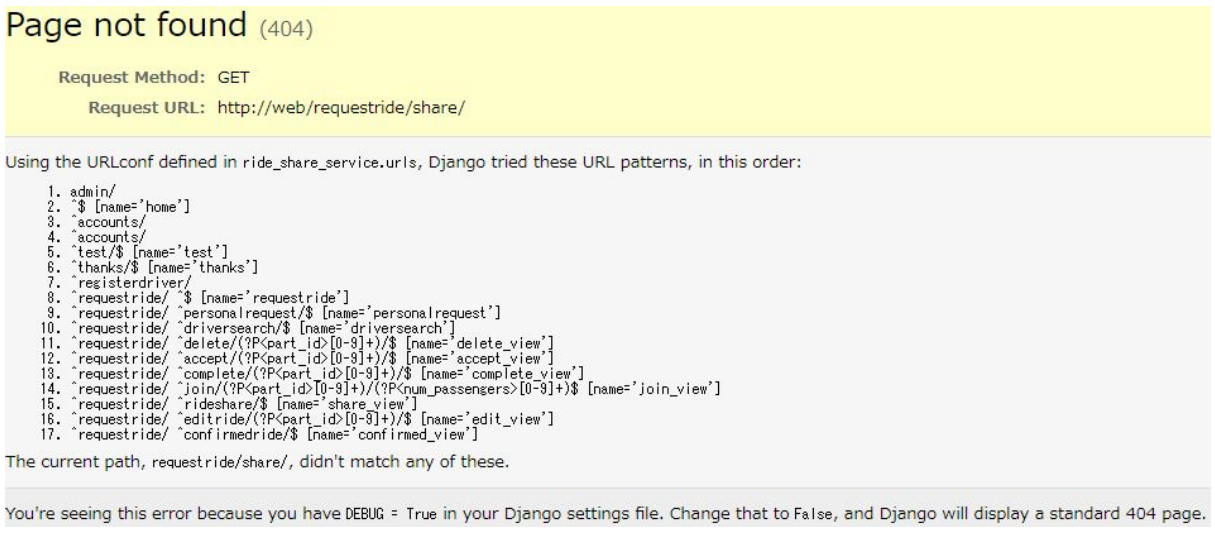
Cannot assign "<SimpleLazyObject: <djang

```
Request Method:      POST
Request URL:         http://web/requestride/
Django Version:      2.1.7
Exception Type:      ValueError
Exception Value:     Cannot assign "<SimpleLazyObject: <django.contri
Exception Location:  /usr/local/lib/python3.7/site-packages/
Python Executable:   /usr/local/bin/python3
Python Version:      3.7.2
Python Path:         ['/code',
                      '/usr/local/lib/python37.zip',
                      '/usr/local/lib/python3.7',
                      '/usr/local/lib/python3.7/lib-dynload',
                      '/usr/local/lib/python3.7/site-packages']
Server time:         Wed, 20 Mar 2019 23:51:13 +0000
```

Figure 6: Test of Login Authentication on request ride page

In Figure 6, we directly enter the URL: http://{addr}/requestride/rideshare/ to get into the left page to request ride. After I click the submit button, the right picture which indicate the ValueError is shown. Obviously, this happen is because of login authentication fallacy.

DEBUG = True problem:


Page not found (404)

```
Request Method:  GET
   Request URL:  http://web/requestride/share/
```

Using the URLconf defined in `ride_share_service.urls`, Django tried these URL patterns, in this order:

```
 1. admin/
 2. ^$ [name='home']
 3. ^accounts/
 4. ^accounts/
 5. ^test/$ [name='test']
 6. ^thanks/$ [name='thanks']
 7. ^registerdriver/
 8. ^requestride/ ^$ [name='requestride']
 9. ^requestride/ ^personalrequest/$ [name='personalrequest']
10. ^requestride/ ^driversearch/$ [name='driversearch']
11. ^requestride/ ^delete/(?P<part_id>[0-9]+)/$ [name='delete_view']
12. ^requestride/ ^accept/(?P<part_id>[0-9]+)/$ [name='accept_view']
13. ^requestride/ ^complete/(?P<part_id>[0-9]+)/$ [name='complete_view']
14. ^requestride/ ^join/(?P<part_id>[0-9]+)/(?P<num_passengers>[0-9]+)$ [name='join_view']
15. ^requestride/ ^rideshare/$ [name='share_view']
16. ^requestride/ ^editride/(?P<part_id>[0-9]+)/$ [name='edit_view']
17. ^requestride/ ^confirmedride/$ [name='confirmed_view']
```

The current path, `requestride/share/`, didn't match any of these.

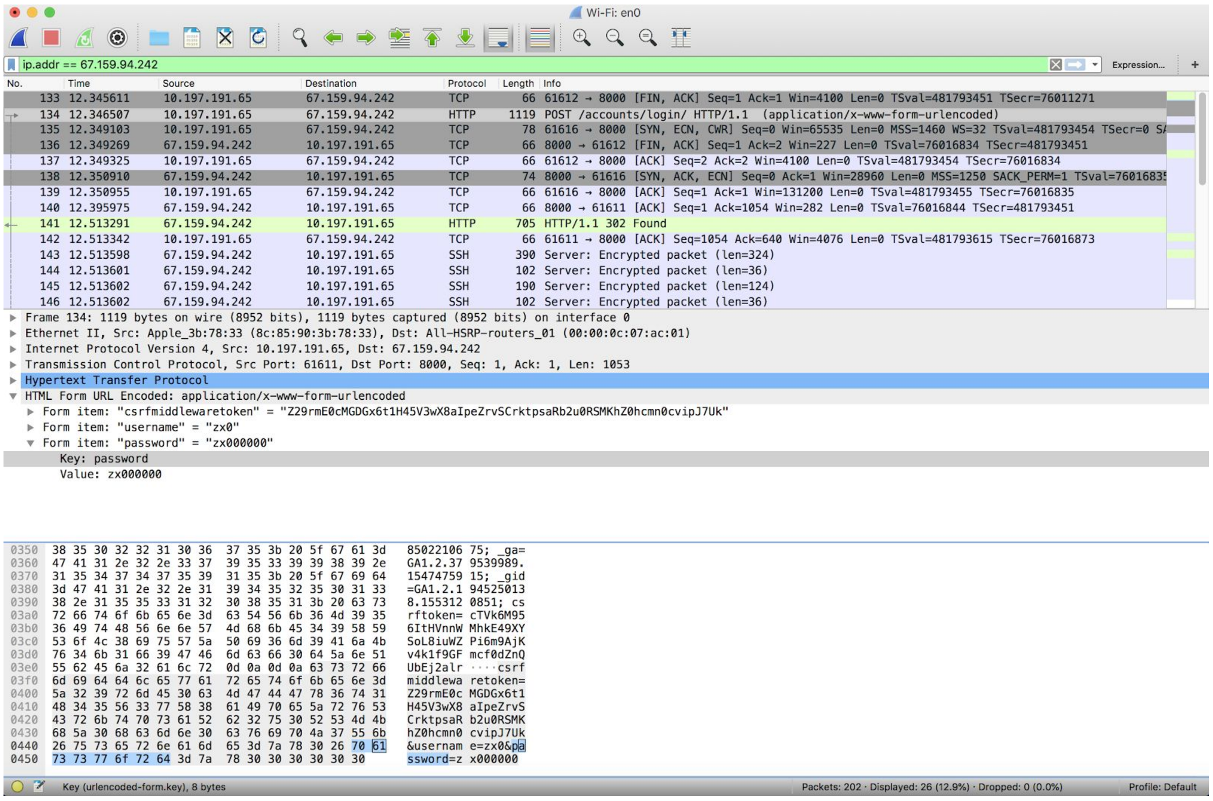
You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

Figure 7: After input of URL doesn't exist

In Figure 7, all URL patterns are displayed to users as a wrong URL is inputted. This is absolutely not correct, and the problem is because it set DEBUG = Ture. If it is change to DEBUG = False, Django will simply return a standerd 404 error page.

HTTP Exploits:

Because HTTP does not encrypt the messages in the transmission, passwords have a risk to be exposed to attackers in the middle. As in figure 8, we captured a user's username and password successfully using Wireshark.



```
Wi-Fi: en0
ip.addr == 67.159.94.242
No.  Time       Source         Destination    Protocol Length Info
133 12.345611  10.197.191.65  67.159.94.242  TCP        66 61612 → 8000 [FIN, ACK] Seq=1 Ack=1 Win=4100 Len=0 TSval=481793451 TSecr=76011271
134 12.346507  10.197.191.65  67.159.94.242  HTTP     1119 POST /accounts/login/ HTTP/1.1  (application/x-www-form-urlencoded)
135 12.349103  10.197.191.65  67.159.94.242  TCP        78 61616 → 8000 [SYN, ECN, CWR] Seq=0 Win=65535 Len=0 MSS=1460 WS=32 TSval=481793454 TSecr=0 SA
136 12.349269  67.159.94.242  10.197.191.65  TCP        66 8000 → 61612 [FIN, ACK] Seq=1 Ack=2 Win=227 Len=0 TSval=76016834 TSecr=481793451
137 12.349325  10.197.191.65  67.159.94.242  TCP        66 61612 → 8000 [ACK] Seq=2 Ack=2 Win=4100 Len=0 TSval=481793454 TSecr=76016834
138 12.350910  67.159.94.242  10.197.191.65  TCP        74 8000 → 61616 [SYN, ACK, ECN] Seq=0 Ack=1 Win=28960 Len=0 MSS=1250 SACK_PERM=1 TSval=7601683
139 12.350955  10.197.191.65  67.159.94.242  TCP        66 61616 → 8000 [ACK] Seq=1 Ack=1 Win=131200 Len=0 TSval=481793455 TSecr=76016835
140 12.395975  67.159.94.242  10.197.191.65  TCP        66 8000 → 61611 [ACK] Seq=1 Ack=1054 Win=282 Len=0 TSval=76016844 TSecr=481793451
141 12.513291  67.159.94.242  10.197.191.65  HTTP      705 HTTP/1.1 302 Found
142 12.513342  10.197.191.65  67.159.94.242  TCP        66 61611 → 8000 [ACK] Seq=1054 Ack=640 Win=4076 Len=0 TSval=481793615 TSecr=76016873
143 12.513598  67.159.94.242  10.197.191.65  SSH       390 Server: Encrypted packet (len=324)
144 12.513601  67.159.94.242  10.197.191.65  SSH       102 Server: Encrypted packet (len=36)
145 12.513602  67.159.94.242  10.197.191.65  SSH       190 Server: Encrypted packet (len=124)
146 12.513602  67.159.94.242  10.197.191.65  SSH       102 Server: Encrypted packet (len=36)
▶ Frame 134: 1119 bytes on wire (8952 bits), 1119 bytes captured (8952 bits) on interface 0
▶ Ethernet II, Src: Apple_3b:78:33 (8c:85:90:3b:78:33), Dst: All-HSRP-routers_01 (00:00:0c:07:ac:01)
▶ Internet Protocol Version 4, Src: 10.197.191.65, Dst: 67.159.94.242
▶ Transmission Control Protocol, Src Port: 61611, Dst Port: 8000, Seq: 1, Ack: 1, Len: 1053
▶ Hypertext Transfer Protocol
▼ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▶ Form item: "csrfmiddlewaretoken" = "Z29rmE0cMGDGx6t1H45V3wX8aIpeZrvSCrktpsaRb2u0RSMKhZ0hcmn0cvipJ7Uk"
  ▶ Form item: "username" = "zx0"
  ▼ Form item: "password" = "zx000000"
      Key: password
      Value: zx000000

0350  38 35 30 32 32 31 30 36  37 35 3b 20 5f 67 61 3d   85022106 75; _ga=
0360  47 41 31 2e 32 2e 33 37  39 35 33 39 39 38 39 2e   GA1.2.37 9539989.
0370  31 35 34 37 34 37 35 39  31 35 3b 20 5f 67 69 64   15474759 15; _gid
0380  3d 47 41 31 2e 32 2e 31  39 34 35 32 35 30 31 33   =GA1.2.1 94525013
0390  38 2e 31 35 35 33 31 32  30 38 35 31 3b 20 63 73   8.155312 0851; cs
03a0  72 66 74 6f 6b 65 6e 3d  63 54 56 6b 36 4d 39 35   rftoken= cTVk6M95
03b0  36 49 74 48 56 6e 6e 57  4d 68 6b 45 34 39 58 59   6ItHVnnW MhkE49XY
03c0  53 6f 4c 38 69 75 57 5a  50 69 36 6d 39 41 6a 4b   SoL8iuWZ Pi6m9AjK
03d0  76 34 6b 31 66 39 47 46  6d 63 66 30 64 5a 6e 51   v4k1f9GF mcf0dZnQ
03e0  55 62 45 6a 32 61 6c 72  0d 0a 0d 0a 63 73 72 66   UbEj2alr ····csrf
03f0  6d 69 64 64 6c 65 77 61  72 65 74 6f 6b 65 6e 3d   middlewa retoken=
0400  5a 32 39 72 6d 45 30 63  4d 47 44 47 78 36 74 31   Z29rmE0c MGDGx6t1
0410  48 34 35 56 33 77 58 38  61 49 70 65 5a 72 76 53   H45V3wX8 aIpeZrvS
0420  43 72 6b 74 70 73 61 52  62 32 75 30 52 53 4d 4b   CrktpsaR b2u0RSMK
0430  68 5a 30 68 63 6d 6e 30  63 76 69 70 4a 37 55 6b   hZ0hcmn0 cvipJ7Uk
0440  26 75 73 65 72 6e 61 6d  65 3d 7a 78 30 26 70 61   &usernam e=zx0&pa
0450  73 73 77 6f 72 64 3d 7a  78 30 30 30 30 30 30      ssword=z x000000

Key (urlencoded-form.key), 8 bytes    Packets: 202 · Displayed: 26 (12.9%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 8: password exposed in Wireshark

HW2:

Quality

Abstraction

Although there exist several classes in the code, it cannot be seen as a pure object-oriented programming project in C++. The main files that build sockets for server and clients, listen to requests and handles concurrent HTTP requests are written with several C functions and global variables. The main logic of the proxy can be improved by dividing it into several classes to manage different functionalities and resource.

There are 3 main three high-level classes serving for proxy's main logic:

(1) The module to parse HTTP requests and responses (the HttpMsg class) makes it easier to get different HTTP headers with good extensibility.

(2) The module to simplify the send and receive process of C sockets (the Socket class) includes the content length check and safe transmission of data through sockets. This helps them reduces much code redundancy in the transmission process.

(3) The logger module implements a LFU cache using singleton design pattern, which provides global access and save much resource. The LFU cache guarantees thread safety with a single lock.

Readability

The project has some very good designs - some basic functionalities of the project are separated from the main proxy logic and are very clear to reuse. These encapsulations are well-designed and have good code readability.

Documentation

Danger log and readme files are empty. No test cases are provided and there is no log outputs when running the proxy. These makes us really hard to read and analyze the code.

Functionality

The docker compose file has a problem with port exposure and log file path. We cannot run the program directly through the docker deployment. We corrected the docker compose file before test.

The log and cache modules appeared in the project folder, but not integrated in the main logic. HTTPS is not supported. Only HTTP GET method is supported. No test cases are provided and there is no log. So, we just tested several existing features.

Security

Although we can send GET requests to websites using HTTP and get the web pages, sometimes it is very slow to receive resource from websites with many pictures and will slow down later responses.

Because there is no cache functionality, the http caching policy will be “no-store” for all responses, which means no responses will be cached and as a result, expiration and re-validation are not needed. And for the cache module that has been implemented but not added to the main logic of the proxy, it does not include cache expiration check and re-validation functions.

Exceptions are raised and processed in all cases that may cause problems. So the proxy will crash when running into a problem and needs restart. Also, RAII is not used with threads. So there exists a risk that some threads may never be ended and therefore cause memory leaks.

The code also lacks check for responses’ status code. There may exist 404, 501 or other unsuccessful responses. However, the code does not cover these cases.

Similar to HW1, the proxy in HW2 also cannot effectively defense DDoS attacks from GoldenEye. Also, due to the drawbacks of HTTP, the proxy can get access to clear-text information in the requests from the clients (usernames, passwords or some other private information can be stolen if the proxy is compromised by an attacker).

3. Feedback

Zi Xiong:

1) HW1

a) We lack some messages to remind our users what has been finished and what caused errors. A reminder system using modal windows to tell users what happened could be used in the front end part, not only to show necessary responses from the back end, but also to give real-time reminders on invalid inputs.

b) Separating the username to display with the real name may a better way to protect users' privacy. However, we only showed user's name to their drivers, which is necessary for ride sharing use and could still protect users' name information.

c) We do not set limits on users' passwords, like the length or whether username can be included in the passwords. This can be a problem when users use very simple passwords. I think more limits and reminders are needed during the process of registration and password reset.

d) We allowed multiple users to share an email. This can be dangerous if attackers use just a few emails to register a lot of amounts using scripts. We should make user's email a unique field in the database.

   For each user, registration and passwords reset should require the authentication via their unique email. And for the change of email, we think two-way authentication may be a solution to better protect users' amount, which can prevent evil users from changing others' emails.

e) We do not require users to type twice during registration. This can be a problem if user type typos in their inputs, which will make them remember the mistake passwords. We should require users to type twice when registering a new amount, and use emails to let users confirm their registration.

f) To prevent attackers cracking passwords by brute force, locking the amount for a certain time after several time's failure may be an acceptable solution.

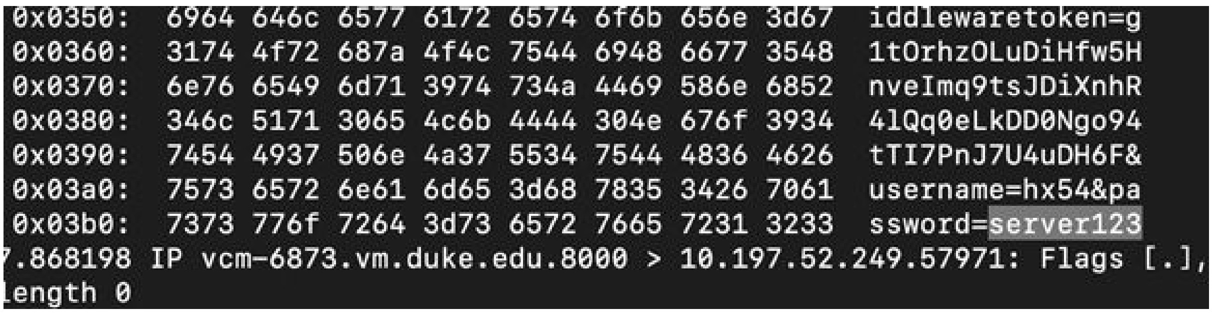
2) HW2

a) According to the reviewers, we do not take HTTP requests of very large size into consideration, for example, some POST requests with a lot of data in the request body. In this case, we should have checked the length of requests and used a loop to send HTTP request.
b) For better performance of the thread-safe cache, read/write lock or some mature library like Redis can be a better choice.

Yunhe Wang:

1) HW1:

We discussed with reviewers about our security problem, and we found that our software is vulnerable to MITM attack, because the application uses HTTP protocol and it does not encrypt the messages in the transmission, so that passwords can be exposed to attackers. Therefore, we can think of deploying the website behind HTTPS to enhance the security.



```
0x0350:  6964 646c 6577 6172 6574 6f6b 656e 3d67  iddlewaretoken=g
0x0360:  3174 4f72 687a 4f4c 7544 6948 6677 3548  1tOrhzOLuDiHfw5H
0x0370:  6e76 6549 6d71 3974 734a 4469 586e 6852  nveImq9tsJDiXnhR
0x0380:  346c 5171 3065 4c6b 4444 304e 676f 3934  4lQq0eLkDD0Ngo94
0x0390:  7454 4937 506e 4a37 5534 7544 4836 4626  tTI7PnJ7U4uDH6F&
0x03a0:  7573 6572 6e61 6d65 3d68 7835 3426 7061  username=hx54&pa
0x03b0:  7373 776f 7264 3d73 6572 7665 7231 3233  ssword=server123
.868198 IP vcm-6873.vm.duke.edu.8000 > 10.197.52.249.57971: Flags [.],
ength 0
```

Our website is reviewed to be immune to major security attacks such as XSS, CSRF. The concern is that we set DEBUG=True which make the website not properly return error, so we should change it to be DEBUG=False.

In addition, our software have some logic issues concerning synchronization and validation. When a driver is about to confirm a ride that apply to him, the ride is shared by another user and the capacity is beyond the number of seat that the driver's vehicle has, and in this case, the server can still accept the ride confirmation request. Therefore, we need to adjust the algorithm to make the server have no ability to accept the ride confirmation as well as inform the driver that the ride has been updated and refresh the page for him.

2) HW2:

We discussed with reviewers about our security problem, and we found some security problems of our software, and we come up with potential solutions to the corresponding problems:

1. Our proxy work well with accessing most of website with CONNECT and POST method but for some websites with GET method, 404 error can occur and the css file cannot be acquired appropriately. We can improve the algorithm of GET method to reinforce the performance of the proxy.

2. Our proxy is vulnerable to DoS attack since it has no implementation of blocking unrelated requests nor a thread pool that limits the number of requests sent at one time, and as more and more requested are sent, the proxy stops working. A way to avoid it is to make the proxy block unsolicited packets, or we can implement a thread pool with limited number of requests permission to limit the usage of resources when lots of requests happen simultaneously.
3. Unauthorized access attack: An attacker can use our proxy to attack other sites. We need to make change to our proxy that only give permissions to certain clients to access the proxy based on different IP addresses, so that malicious users cannot utilize our proxy.

4. Resources

[1] We used an open-source DDoS attack test tool called GoldenEye from https://github.com/jseidl/GoldenEye.

[2] We used Wireshark to capture HTTP data frames during its transmission.