Duke University
ECE568 Homework 3:
Security

Ruihan Xu (rx29)

Rijish Ganguly (rg239)

Date of submission: 03/20/2019

SECTION 1: CODE QUALITY

1. Homework One

In this section I'm going to discuss the program written by yt114 and wy48 for the Django website application.

1. The code for Homework One appeared to be well abstracted. The Objected Oriented Programming paradigm was followed accurately to design the web-application. Each requirement was broken down into specific classes with methods.
2. The code wasn't commented. The absence of comments made it a little difficult to understand the utility and the reasoning behind using a particular function.
3. The code is easy to read and understand, even though comments are absent. The workflow is simple and can be comprehended by amateur programmers.
4. The code has necessary error checking and exception handling, but the error-handling is not exhaustive. The HTML pages are simple and extends from a base template, hence making the entire website uniform. However, the app is not responsive and lacks necessary margins and spacing between HTML elements.
5. Code wasn't repeated and I was able to infer how exactly the web application works. The login module defined a user class for registration. The ride module defines another two classes Share and Ride for handling the booking and sharing of a ride.
6. Overall, the code wasn't well documented as there wasn't any explanation behind the working and the usage of the web-application through a README file. The commit messages weren't useful enough to decipher what was being committed.
7. The code uses docker container to run.

2. Homework Two

The homework two, developed by J.C and C.J did an over-all great job.

First, they have 2 cpp file and 2 header files, which is good for a project with a medium amount of code. They have dedicated a log header to deal with log file I/O and a proxyserver header to handle the requests and responds. Insead of the proxyserver header, they defined three classes, HttpRequest, client_info, and HttpResponse. A small inconsistency here is to name client_info instead of ClientInfo, which is what we normally name a Class. In the proxyserver, they defined several functions like send_message_to_server(). Since they already have a header file, I believe it is better to move those functions in the header. Also, we normally use underscores in variable names only. This suggestion applies to both the log head and some functions in proxyserver.

Second, moving on to the detailed code inside the log header and file. Since main functions call them a lot, this can largely reduce repeated code. The use of lock_guard<mutex> is a great idea that follows RAII and provides multi-thread support. But I think the naming can be more explicit here. The client_info class constructor did not use the initialization list. The log_request will print a redundant '\n' because gettime() will return a string ended with '\n'.

Third, the proxyserver part. Apparently, they did not implement a daemon version and they print "running" to stdout. This is not satisfying the requirements. In the receive_data_from_server function, they used pthread_mutex_lock to lock the receive buffer. Apart from the fact that it is better to use mutex::lock together with stl thread, I do not see why they implemented a mutex here. From my understanding, the only two places that concurrency happens is at the cache part and the log part. They did a good job using lock_guard<mutex> at log but they did not implement the cache. Speaking of which, in the HttpResponse class, there is a field to deal with the cache. Since they were not used, it

is better to delete them. In receive_data_from_server, the buffer size is set to be 60000(MAXSIZE) bytes, which is likely to be larger than most GET response. Their receiver implementation actually supports a much smaller buffer size. Reducing the size can save some cost. What is more, MAXSIZE is used as the upper limit of the thread number. This is not ideal since it will be hard to change one of the values. At line 218, string clininfo is not used after constructed.

Overall, they did a good job in abstraction. Connecting, sending and receiving are frequently used in all three methods and abstract them largely reduced repeated code. However, they barely have any documentation. The useless commented codes were not deleted. The code is still readable since the variable names and function names are concise and accurate. I can follow most part of their code without any problems because I have done that homework as well. But this may be quite confusing for other people without prior experience without documentation. For example, the while loop they used to receive data from the server.

SECTION 2: SECURITY

1. Homework One

The Django framework provides a lot of security against a few known attacks. As the code uses Django templates, it's protected against Cross-Site Scripting. Django templates escape certain specific characters which are dangerous to HTML. The code maybe protected from SQL injection because a query's SQL code is defined separately from query parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver. The web application is also protected from CSRF attacks as the code for the web-application uses CSRF tokens for POST requests. The protection enabled works by checking for a secret in each POST request. This ensures that a malicious user cannot simply "replay" a form POST to the website and have another logged in user unwittingly submit that form. Django by default uses PBKDF2 and hence the web-application is resilient to brute-force attacks and rainbow table attacks. The web application uses common password validators such as minimum-

length, common password and numeric password validator. The website application is protected by clickjacking by the X-Frame-options middleware which in a supporting browser prevents a site from being rendered inside a frame.

The web-site application is secure against a format string injection attack as we are using a python framework and the code doesn't use format string parameters. The program is also safe against command injection as it doesn't execute user input arguments.
In my opinion, we can still carry out a multitude of attacks against the website application such as the following:

A. DoS – Denial of Service attack

1) Ping of death is DoS attack caused by an attacker deliberately sending an IP packet larger than 65536 bytes allowed by the IP protocol. Many Operating systems don't know what to do when they receive oversized packets, so they freeze or crash. When a packet is sent from a source computer to a target machine, the ping packet gets fragmented into smaller groups of packets. When these packets reach the target computer, they arrive in fragments. So, the target computer reassembles the malformed packets which are received in chunks. But, the whole assembled packet causes buffer overflow at the target computer. I wrote a bash script in order to carry out this attack:

   while true; do
   sudo ping 152.3.52.7 -l 65500
   done

   Then I gave sudo privilege to this bash script and executed it. Although I expected this attack to succeed, it was met with failure. The performance wasn't affected drastically. The website application was working properly.

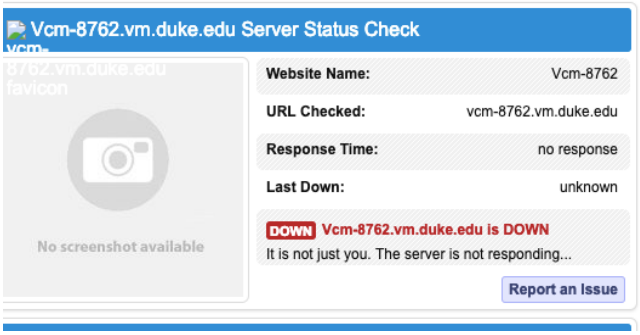
2) DoS with TorsHammer: I downloaded the TorsHammer project from Github. TorsHammer is a slow HTTP POST DoS tool , meaning it can possibly bring a

webserver down using maliciously crafted HTTP requests even with a single machine without much bandwidth. It mostly works with apache and IIS and can be used over TOR, hiding real attacker. Torshammer executes a DoS attack by using a slow POST attack, where POST content is transmitted in slow rates under the same session (actual rates are randomly chosen within the limit of 0.1-3 seconds). It will generate number of HTTP POST requests and establish the connection to the server If server can't close the connection correctly, it will get a lot of current connection at same time and lot of child processes/thread will be spawned. Inside of each of connection, it is just only sending some randomize characters to maintain the established connection. When the attack hits the maximum of child process/thread that server can open, it will no longer serve the legitimate traffic and will become unavailable.

I execute the program as following:

./torshammer.py -t 152.3.52.7

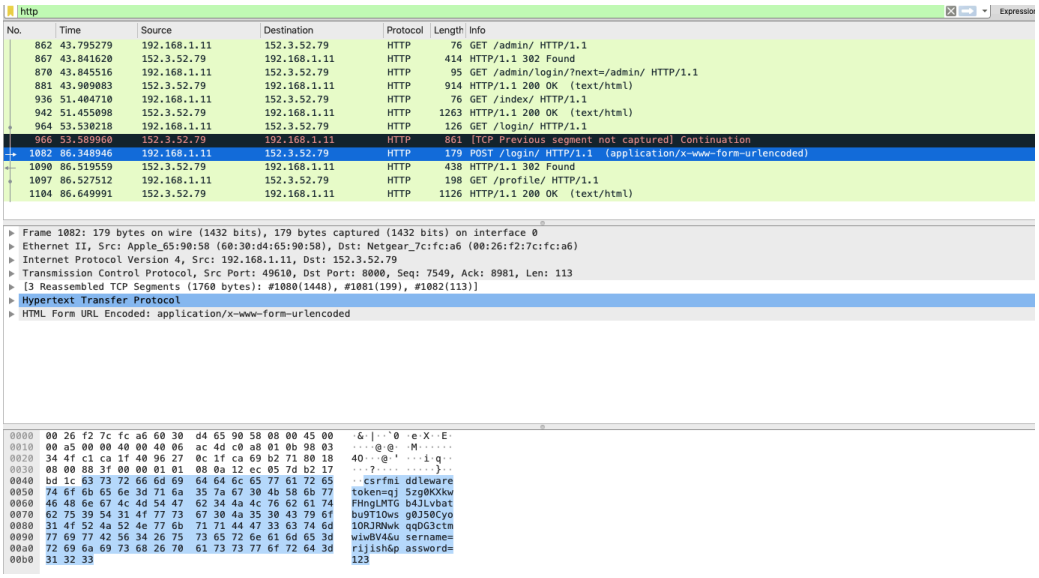
The server goes down in a matter of seconds.



Hence, I was successful in carrying out a DoS attack.

B. Absence of SSL

The website application isn't deployed behind HTTPS. Without SSL, it is possible for malicious network sniffers to sniff authentication credentials thus compromising the credibility and integrity of login data. As a result of this, I was able to sniff login credentials when I tried to login to the web application using WireShark. The username and password were available to me in plain-text.



Even if SSL is enabled, it is possible to decrypt encrypted traffic. Firefox and Chrome both support logging the symmetric session key used to encrypt TLS traffic to a file. You can then point Wireshark at said file and get access to decrypted TLS traffic.

C. Password Protection and 403 page

There is no restriction imposed on password length and username. It's important to have passwords with length greater than eight. The authentication method used also doesn't check whether the password or username has special characters in it or uses common words. If a user is not registered as driver and tries to search for rides as a driver, a 403-error page is shown. It's better to have a page explaining the limited access to the "search for rides as a driver page" than to have a 403-error page. You need to either make your own error pages or edit the default error pages in a way that hides any information that could be used by an attacker. Default 400 or 500 pages often include a lot of information about the error, and this is very useful for an attacker - they can find out what software, OS and web-server is running etc.

2. Homework Two

First of all, the program does not validate the user input sufficiently. The proxy_port may be out of range or already occupied. Say there is another process using port 12345, the proxy will exit without explicitly print out the cause of the error.

At line 297, they did not check the return value of setsockopt(). Chances are that setsockopt may fail if the socketfd is not a valid descriptor or some addresses are invalid. Their code will not be able to deal with that condition. Furthermore, most of the code does check the return value of the functions but all the code do is cerr and exit. An ideal proxy should be able to run even if there are errors. Instead of exit, it should log the error and try it best to deal with the error. It will be great if they can use exception handling instead of return -1.

In functions including connect_server and send_message_to_server, the group threw a runtime error. It is generally a good idea to use exception handling but they did not attempt to catch the exception. The programming will terminate.

Apart from those vulnerabilities, some can be generated to attack the proxy:

A. Thread Overflow

   At line 337, I noticed that this group simply used a multithreading solution with an upper limit of 60,000 threads. We can construct a very simple attack to shut the server down. Send a HTTP GET request 60000 times and the program will automatically stop.

To attack, simply type in terminal:

for n in {1..60000}; do

cat requests | nc <proxyIP> 12345 > resp.txt - | nc example.com 80;

done

The requests file contains:

GET http://example.com/ HTTP/1.1

Host: example.com

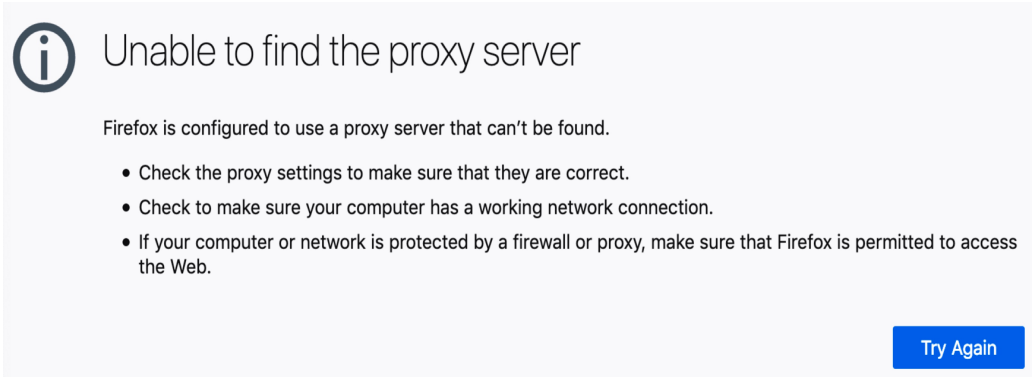
After a short while, the proxy exited.

Practically, the browser tends to send lots of requests than we think. A simple HTTP CONNECT can initialize 10 or more threads. My suggestion to solve this problem is by using a thread pool. Although thread pool is not supported by stl yet, there are implemented code on GitHub. A thread pool ensures that once a thread exits, another new thread can be created.

B. Memory Corruption Attack

In function connect_server, the use of free() can have some problems when trying to access website like http://www.cplusplus.com/. I ran their docker on my virtual machine and every time I tried to access this website, this error happens: free(): invalid next size (normal). I tried to figure out the reason but cannot give a promised result. The function free() is not what we want in C++. The problem might be solved by using smart pointers.

To attack, simply type http://www.cplusplus.com/ in the Firefox browser two times and the proxy will be off-line.



C. Malicious Website Attack

At line 134 in proxyserver.h, the parser function will exit -1 if the response does not contain "\r\n". This will make the proxy very vulnerable to attacks. Attackers can use Netcat or similar programs to deliberately build a server that can return a response without "/r/n". Then the attacker can use the proxy to access that server and get a mal-formatted response. The proxy will exit when this happens.

Notice that the single buffer has a size of 60MB, which means every thread will at least takes 60MB of memory. The attacker can build a website that has lots of files or sends lots of cookies. The proxy will try to GET all of them at the same time, which means a great amount of memory space may be occupied and some of the vector<char> may not be allocated. Errors may happen or the speed of the proxy will become very slow.

SECTION 3: LESSONS LEARNED

1. **Ruihan Xu(rx29)**

   Reviewing other people's code as well as getting feedback from our peer has benefited me a lot. This part will focus on the lessons learned and the potential improvement of my code.

   A. Homework One

   The code quality of my homework1 is abstracted, readable and clear according to the group reviewed my code. However, they also pointed out that the documentation is not detailed enough. In forms.py and models.py, I barely wrote any documentation. After learning Django, the code appears to be very straight forward to me. But I ignored the fact that it is one thing to follow your own naming convention and another to follow others. I should be more explanatory in the future of my development career.

   The security mechanism for Django is already well designed. Things like the login credentials are handled gracefully. The main problem that I did not deal with is to check the integrity of the user input. Users can input mal-formatted fields and the whole website can be brought offline because of that. For example, the expected arrival time can be a time point in the past. The passenger number can be 0 and still request a ride. Those details need to be taken care of otherwise lots of problems may happen. Another issue is the protection of the data. As long as the user finds out the URL pattern, it is possible for them to modify the data that they are not supposed to modify anymore. For example, user A can modify User B 's data if he knows the link

pattern and the ride id. The lack of data protection and user authentication can be very dangerous. For example, other user can alter the details of your requested ride in Uber. This may lead to a loss of money and time. More seriously, they can take you to some places no one knows and change the destination to other places. From a bigger view, lack of data protection and authentication can literally break the whole world down. Hackers can change the value of stocks, crash an airplane, or even launch a nuclear weapons. As a Django beginner, I should spend more effort on the security of my website in the future.

B. Homework Two

Regarding code style, the group reviewed our code gave a positive feedback. The naming was clear, which I paid attention to during coding; The comments are clear and the documentations including the danger log are well structured and explanatory. But I do not think we did a great job in abstraction. Unlike the Django homework, where the framework itself is already abstracted, we started homework 2 from scratch. We were more anxious about the detail implementation of the proxy instead of the abstraction. We started coding without planning beforehand. Putting pieces together was a tough job, not to mention that we did not utilize good abstraction. However, due to the limited time, we decided to move on and put everything in a single file. Form the feedback as well as my personal experience in homework 2, I have learned the importance of abstraction. Good abstraction can not only make the code more readable, but also reduce redundant code. For example, we wrote repeated code to connect to the server in GET, POST and CONNECT, which should be replaced by a connect function. Also, all the code was crowded together in a single file. This may be easy for developing a small project like homework2 but will be very troublesome to maintain. For example, if we want to upgrade the proxy or add more functions to it, bad abstraction can be a real pain. Plan before implement was what we have learned in ECE550 and we should keep it in mind.

Comparing to homework1, which is based on the well-designed web framework Django, homework2 appears to be more flexible and is thus prone to attacks. The code from another group that I was reviewing did not use the thread pool, which leads to the most common attack that can happen: the thread overflow attack. We did a good job in solving this problem by using a thread pool. Thread pool is not in STL now but is widely used by programmers. Thanks to the open-source policy that enabled us to get the code of thread pool from GitHub.

But we do have some flaws. The group checking our code told us that the proxy was prone to cookie leakage, as the design of cache was not specific enough. Without adequate information, the proxy is likely to access the server more than we thought. If the server sends lots of cookies, the proxy may be very slow. Also, what they did not notice is that our proxy has some issue accessing certain website like cplusplus.com. The performance was not stable and sometimes the proxy exits unexpectedly. By studying ECE568 and digging into this assignment, I realized the difficulty of writing a resilient, secure and error-free server software.

2. **Rijish Ganguly (rg239)**

Reviewing other people's code as well as getting feedback from our peers has benefited me a lot and helped me understand about potential vulnerability in my code. This part of the assignment will focus on the lessons learned and the potential improvement of my code.

A. Homework One:

a. According to my peers, there might be SQL injection attacks against our code as we don't check username and password format. This potential error is easy to resolve as we can impose restrictions and prevent use of special characters in username and password combinations. We can also impose restrictions on password length and prevent common words found in the dictionary to be used as passwords. The Django framework provides libraries which can prevent brute-force and dictionary attacks. We can use libraries such as

Django Axes and Django Defender. We can also implement two-factor authentication using Django-Two-Factor-Auth which would make authentication more secure.

b. The second problem pointed out to me was a web-direction problem. According to my peers, if a user logs out, they can still access the dashboard page, which might leady to security issues. The solution to this would be creation of a custom error page which would explain that since the user is not logged in, they don't have access to the dashboard.

Apart from this, we got mostly positive feedback regarding code abstraction, code repetition and code readability. However, there are still a few things which we can improve. There are a few logic flaws such as our app gives a user permission to create a ride before current time and as a result, this ride would never get confirmed. One potential solution would be reminding the user the current time when they try to create a ride with expired time.

In the future, I plan to learn more about security issues and various Django libraries that would help me write secure and robust software.

B. Homework Two

a. The proxy cannot open chunked website using GET Method. Our code only supports reading the content-header field. We need some additional code in the receiver part of proxy that forward requests to support this requirement.

b. Our code had no exception guarantee. This problem can be resolved by more extensive error checking and utilizing try and catch to ensure basic exception guarantee.

c. Revalidation and expiration weren't completed properly. This is a security issue as the cached data could be expired and the proxy doesn't send a message to revalidate. In order to defend against this, we would implement a function that, before responding from the cache, asks the origin server

whether the current version of the data is valid, and if not requests a new version.

d. Our program may be prone to DoS attack. We can utilize a pool of threads with a limitation on the maximum number of threads which can be spawned. Then we should store the incoming requests in a queue data structure and block incoming connections after the queue reaches a certain size.
e. Although mutex was used, there is a race condition when we try to generate unique IDs. Need to explore further to figure out how this race condition is generated. Also, the program needs to be setup in docker, which requires rewriting the daemon code.

According to peer feedback, our program has good OOP practice and RAII was used. However, we should have provided stronger exception guarantee. Although, we were able implement almost all the requirements, our proxy was not immune to external failures in certain cases. We also failed to set up docker, which was a requirement for the project. I feel, we should have done more exhaustive testing in order to tackle a few of the security concerns.