CS 483/5583 Software Security

Assignment #2

This assignment is based on the SimpleWebServer program discussed in class. If you choose to use a different programming language, it is your responsibility to translate the Java code correctly. Turn in your answers and source code as **a single zip** file.

1. What if a client connects to SimpleWebServer, but never sends any data or disconnects? What type of attack would such a client be able to conduct? (10 points)

The attacker could conduct a Denial of Service or DoS attack. This is because whenever a connection request is received, the server creates a new socket and waits for user input from the client. Because of this, if the client never enters any data or disconnects, the socket on the server is left waiting forever for input and occupies that particular socket. If conducted at scale or with multiple requests, this could take up all sockets in the server leading to DoS for other clients who need to access the server.

To solve this, a timeout for the request could be implemented in the run method to timeout and disconnect sockets if they take a long amount of time.

1. Modify SimpleWebServer.java to allow the client to upload files through a PUT command (e.g., PUT <destination\_path> <file\_content>) and save all client requests into a log file. Also modify SimpleWebClient.java to allow the user to upload a given file to the destination path on the server. The sample methods for text file storage and logging are given below. You may update storeFile to deal with binary files if needed. (30 points)

Something else to note is that I modified storeFile function slightly to remove the need for the scanner and read in lines directly from the Buffered Reader. It just seemed redundant to me to pass in the buffer to the scanner which will do the same thing.

See source code (on github) for updated SimpleWebServer.java and SimpleWebClient.java files.

1. Rewrite the serveFile method such that it imposes a maximum file size limit. If a user attempts to download a file larger than the maximum allowed size, write a log entry to a file called error\_log.txt and return a “403 Forbidden” HTTP response code. (20 points).

See source code (on github) for updated SimpleWebServer.java.

1. Describe and implement: (a) an attack that defaces (overwrites) the index.html homepage and (b) an attack that removes the log data. Attach screenshots of your attacks. (20 points)
2. Since in #2 we added the ability to PUT files, the ability to overwrite/deface the index.html page is quite simple. All the attacker must know is where the index.html file is located on the server, perform a GET request to get the existing index.html file, and upload a new file using the PUT endpoint. The only requirements are that it must be in the same location as the index.html file, and have the same name (index.html). From here, the server will save the new index.html file to the directory, overwriting the existing file.

This allows us to return the existing index.html and get the file contents, modify it to how we see fit, and then reupload it to the server. Below are screenshots of what the attack would look like:

NOTE: in the PUT request the first index.html is the original file, and the second index.html is the new updated file

GET /

PUT index.html index.html

Text

Description automatically generatedText

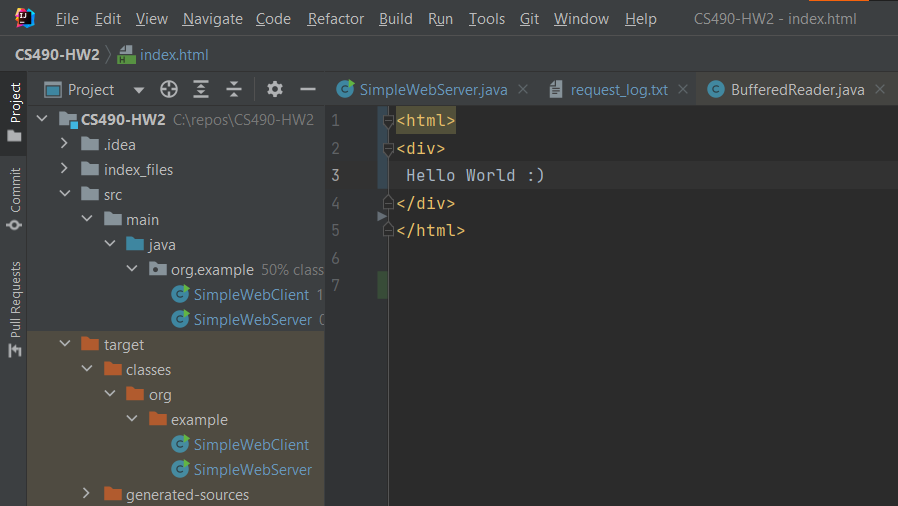
Description automatically generated

File contents before attack:

Text

Description automatically generated

Contents after attack:



New index.hmtl file from client is now uploaded to server.

1. To then remove one of the log files, the general process is quite similar. We would again need to know the location of the log file we are trying to remove data from, as well as the name of the log file. However, once we know that we can send a PUT request to the server with a new file that is empty, and have the server save our new file. The attack would look something like this:

NOTE: the first error\_log.txt is the original file, and the second error\_log.txt is our new empty file

PUT error\_log.txt error\_log.txt

Text

Description automatically generated

From this, new empty error\_log.txt file is uploaded to the server.

File contents before attack:

Text

Description automatically generated

File contents after attack:

Graphical user interface, text

Description automatically generated

1. Suppose you are the attacker who has got hold of the complied SimpleWebServer.class. Describe an attack such that, after SimpleWebServer is re-started (e.g., because of an exception by another attack), the functionality in (3) is disabled. (20 points)

For this attack, it would be important for the attacker to know that our program is using Java (so they know how source code is compiled), as well as where the compiled versions of the source code are located on the server.

Something to note is that I used IntelliJ to develop my program, so the project directory/structure will look slightly different than if I had used Eclipse to develop my program. I make this distinction because the compiled .class files will be in a different location.

With that out of the way, how this attack would work is the attacker would use the GET and PUT request to download and upload files to the server. However, this time we will be downloading/uploading the compiled .class files instead of having the server need to compile them. For our attack, the order of operations are as such:

1. GET /target/classes/org/example/SimpleWebServer.class
2. Attacker modifies the .class file to remove the check added in (3)
3. PUT /target/classes/org/example/SimpleWebServer.class SimpleWebServer.class
4. Attacker restarts server so new .class versions are loaded into memory and are run. Because of this, the functionality in (3) is disabled