Software Architecture - CSSE 477

Milestone 3 - Application Server

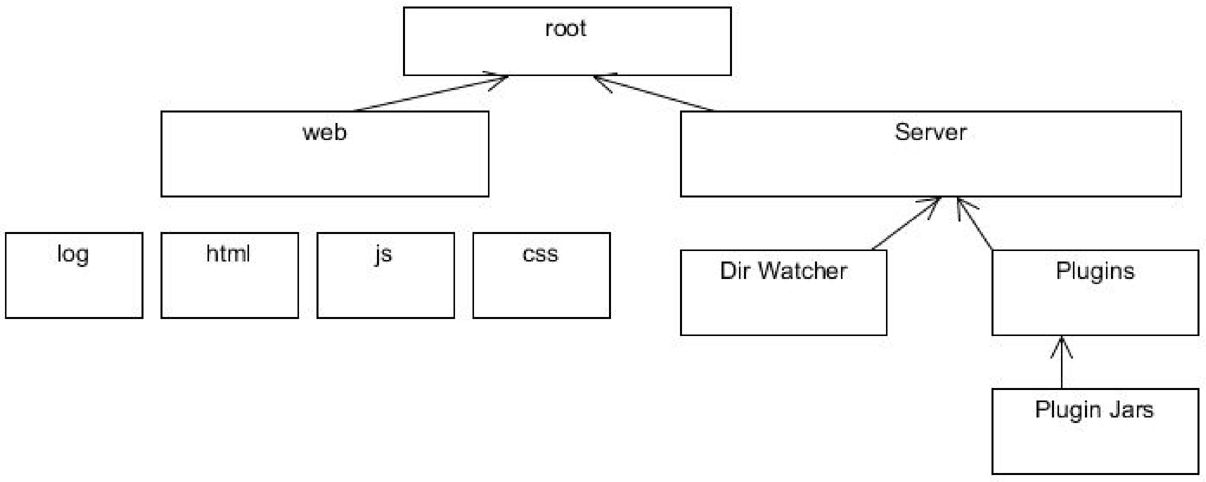
By: Matt Rocco and Paul Bliudzius

Change History

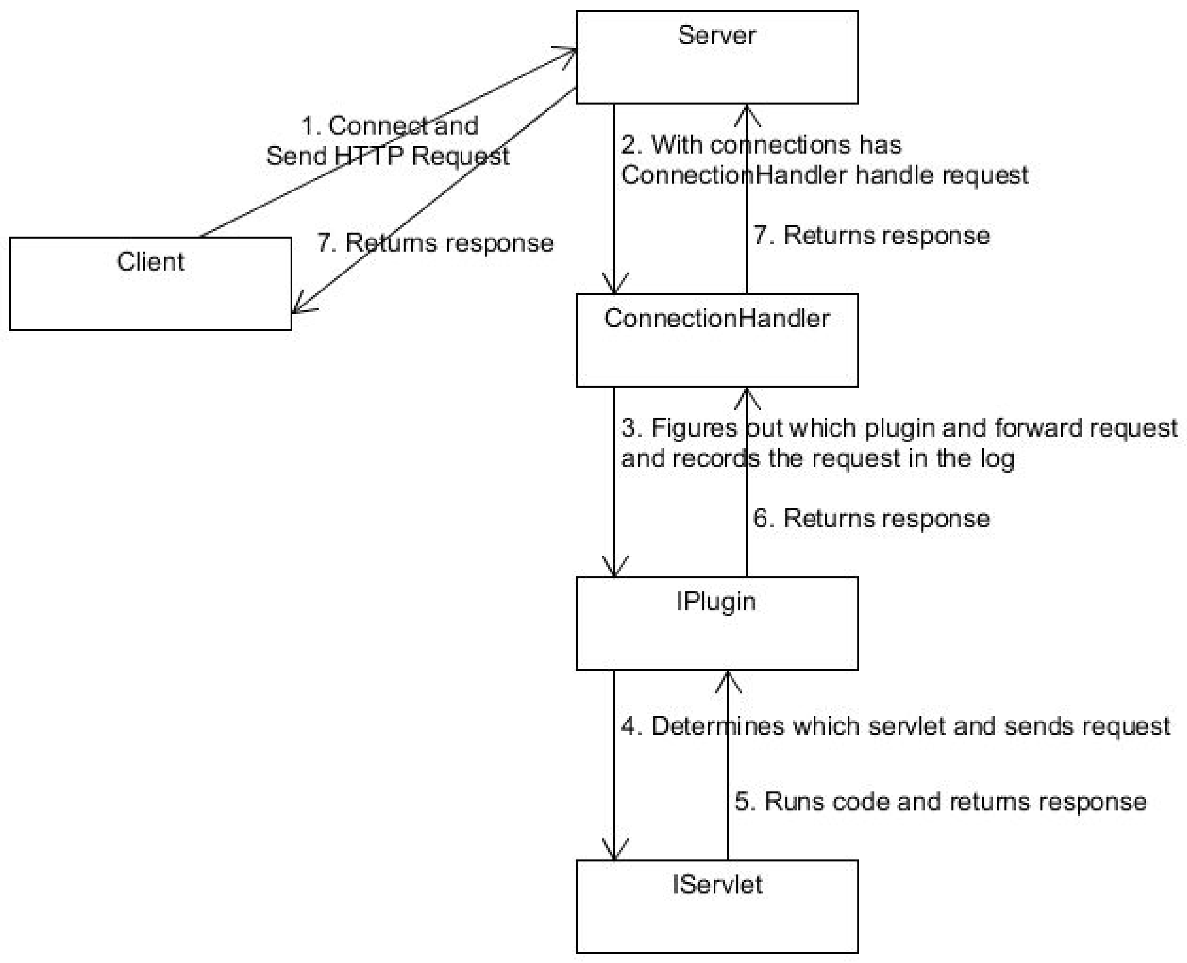
1. We included the ability in the system to use JAR files to allow dynamic loading onto the server to use the JARs to that way there is no need for compiling in previous system of having the java classing dropped in.
2. We have removed the Test Report that was in Milestone 2 since it was for developing for Milestone 2 and no longer relevant because it had been tested, proved working and no longer important to Milestone 3. Instead it had been replaced with Architectural Evaluation and Improvements that cover Available, Performance, and Security for our web server.
3. The feature listing now includes tactics that we implemented into our server.

Architecture and Design

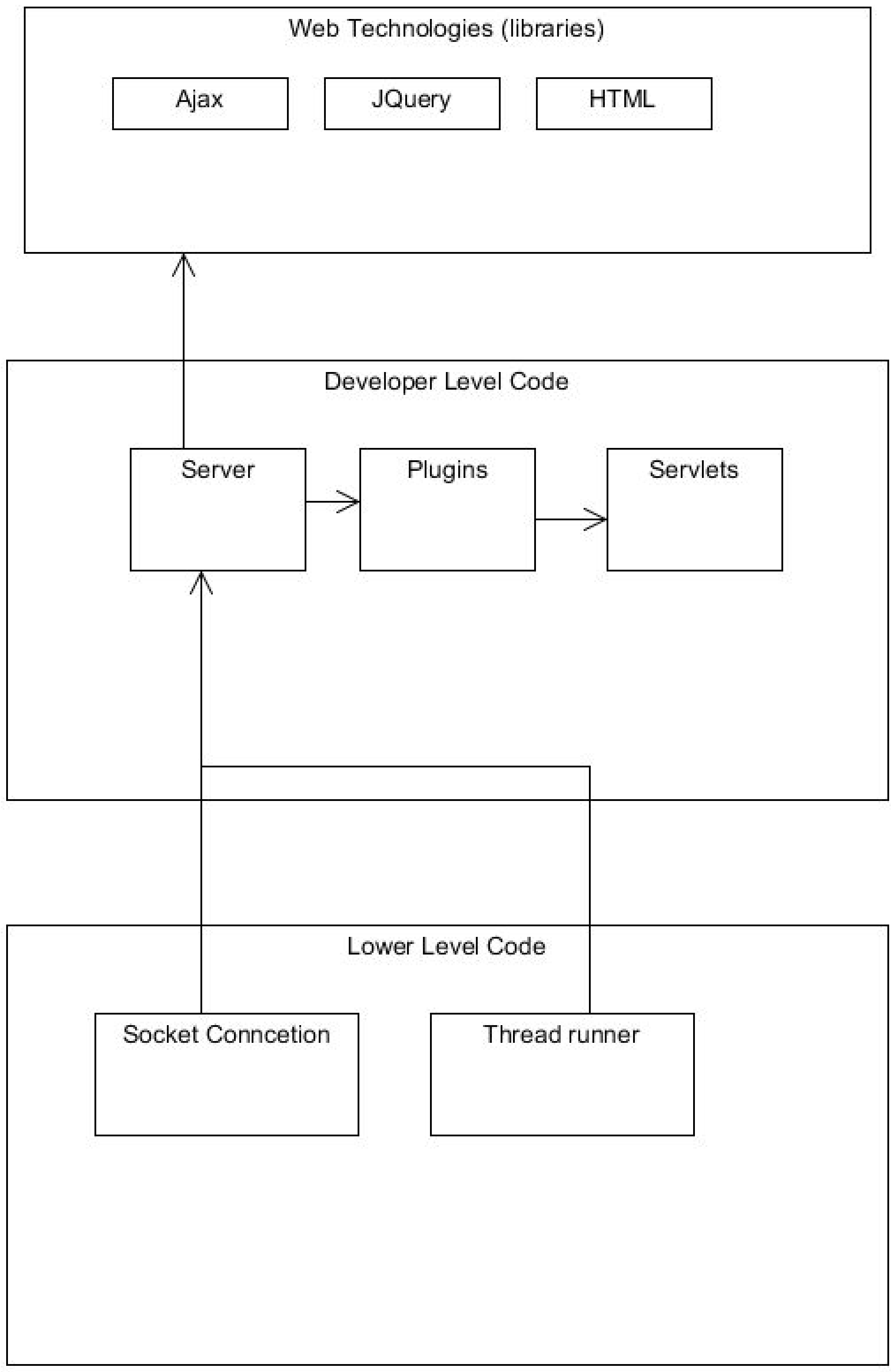
Allocation



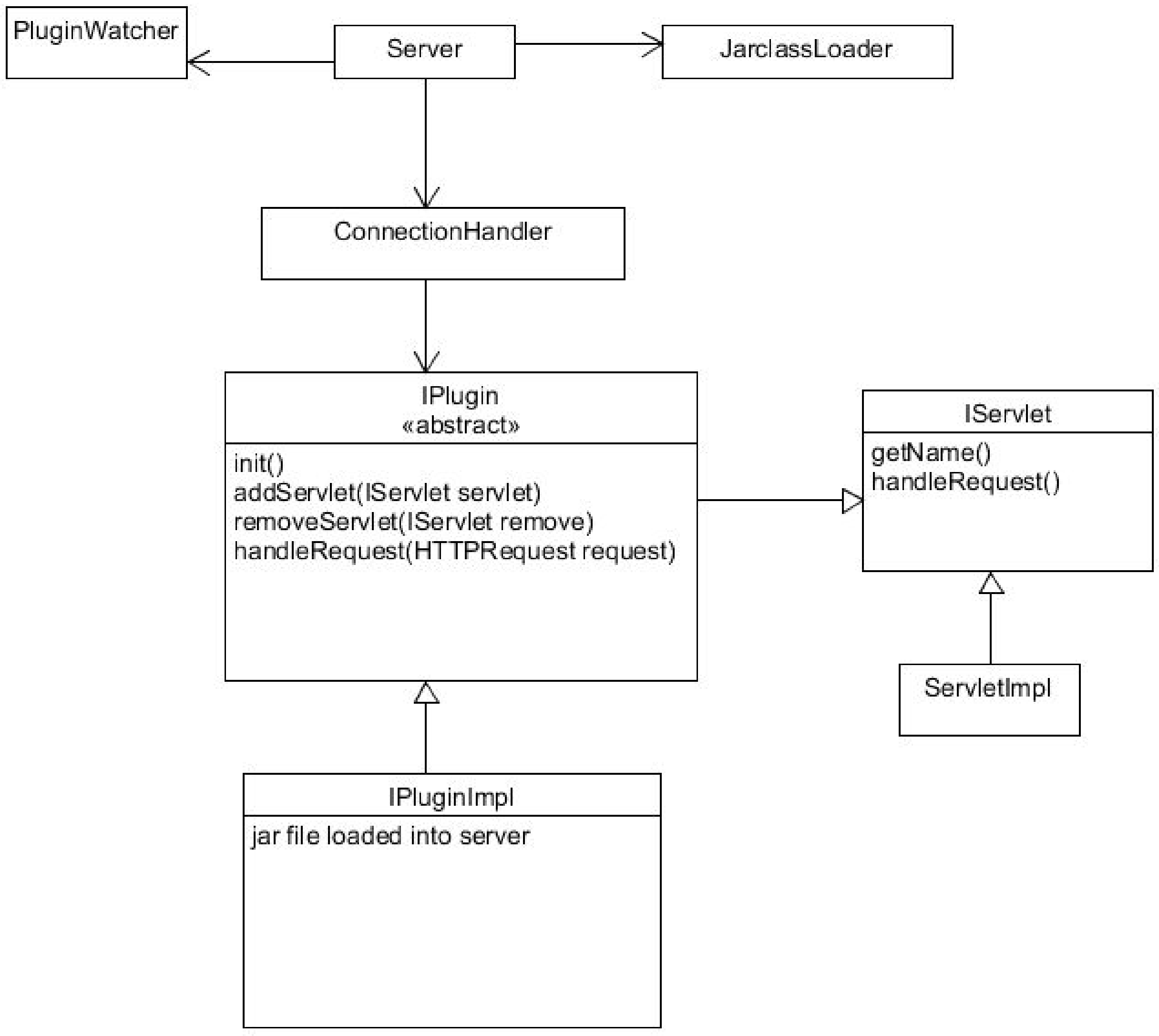
C&C


Server
1. Connect and Send HTTP Request
7. Returns response
Client
2. With connections has ConnectionHandler handle request
7. Returns response
ConnectionHandler
3. Figures out which plugin and forward request and records the request in the log
6. Returns response
IPlugin
4. Determines which servlet and sends request
5. Runs code and returns response
IServlet

Module


Web Technologies (libraries)
Ajax
JQuery
HTML
Developer Level Code
Server
Plugins
Servlets
Lower Level Code
Socket Conncetion
Thread runner

UML



Tactics and Feature Listing

Tactic and Features	Team Member
Servlets	Paul
Directory Watching	Matt
Architecture Design	Matt and Paul
Jar -> Plugin	Matt

Plugin -> Jar	Matt
Plugin Strategy	Paul
Fault Prevention	Paul
DDos Prevention	Matt
Logging	Matt
Resource Control	Paul

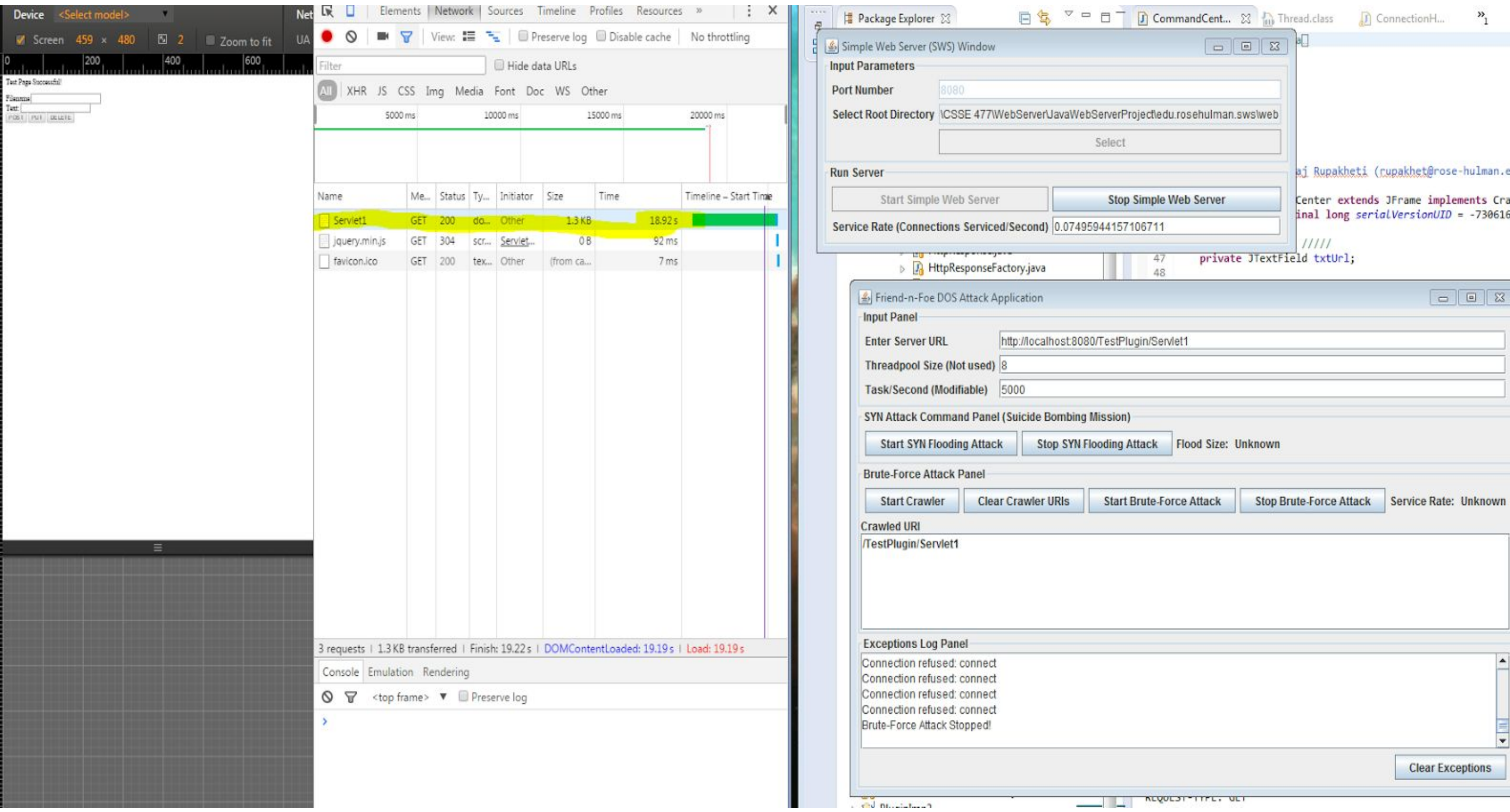
Architectural Evaluation and Improvements

1. **Availability**
   a. DDoS Attack Prevention
      i.

Source	Stimulus	Environment	Artifact	Response	Response Measure
DDoS Server	Numerous HTTP Requests from same server	DDoS Attack	HTTP Server	DDoS is banned and normal users can make calls to server	User can make HTTP Request and receives a response within normal 5 seconds of before DDoS started

ii. Test the server before DDoS is being ran for a time until response. Then run the DDoS for a while and try to get a response within 5 seconds to know that the server is still available.

iii. The image below shows a DDOS attack running against our unprotected server. With 5000 attacks per second, it takes the other client 18.92s to get a response from our server. We need to prevent this!



Name	Me...	Status	Ty...	Initiator	Size	Time
Servlet1	GET	200	do...	Other	1.3 KB	18.92 s
jquery.min.js	GET	304	scr...	Servlet...	0 B	92 ms
favicon.ico	GET	200	tex...	Other	(from ca...	7 ms

3 requests | 1.3 KB transferred | Finish: 19.22 s | DOMContentLoaded: 19.19 s | Load: 19.19 s

Simple Web Server (SWS) Window

Input Parameters

Port Number: 8080

Select Root Directory: \CSSE 477\WebServer\JavaWebServerProject\edu.rosehulman.sws\web

Run Server

Service Rate (Connections Serviced/Second): 0.07495944157106711

Friend-n-Foe DOS Attack Application

Input Panel

Enter Server URL: http://localhost:8080/TestPlugin/Servlet1

Threadpool Size (Not used): 8

Task/Second (Modifiable): 5000

SYN Attack Command Panel (Suicide Bombing Mission)

Flood Size: Unknown

Brute-Force Attack Panel

Service Rate: Unknown

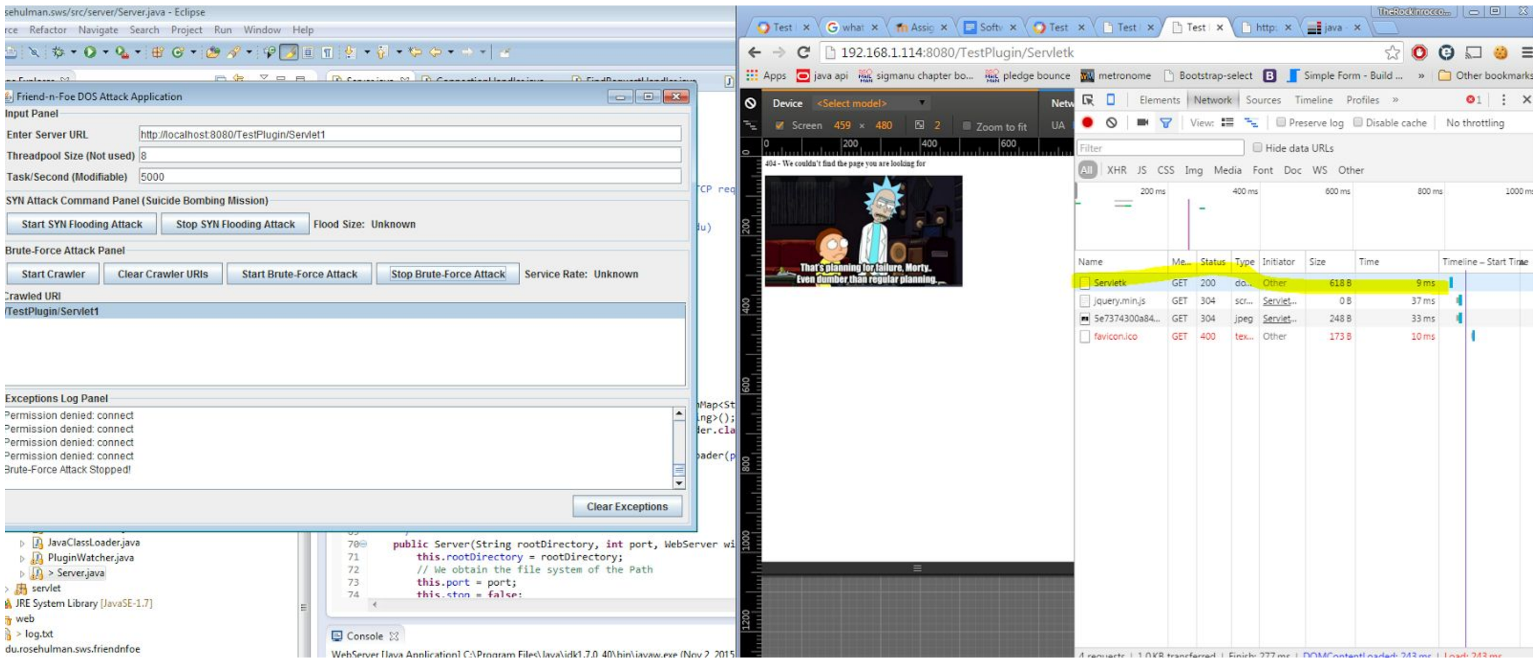
Crawled URI

/TestPlugin/Servlet1

Exceptions Log Panel

```
Connection refused: connect
Connection refused: connect
Connection refused: connect
Connection refused: connect
Brute-Force Attack Stopped!
```

iv. This was implemented by having a counter for each address. If the number of requests from that address surpasses our set limit, then the address is banned for a period of time. This is using a hashmap for the address, an incremented counter, and a limit. Then we also have reset clock that clears that hashmap after an interval. This prevents DDoS because we can track the IP and if it passes the certain number, we will be able to put the address on the ban list. Locking other individuals from being able to access the system, therefore the server not being available to use.



Friend-n-Foe DOS Attack Application

Input Panel

Enter Server URL: http://localhost:8080/TestPlugin/Servlet1
Threadpool Size (Not used): 8
Task/Second (Modifiable): 5000

SYN Attack Command Panel (Suicide Bombing Mission)

Flood Size: Unknown

Brute-Force Attack Panel

Service Rate: Unknown

Crawled URI

/TestPlugin/Servlet1

Exceptions Log Panel

```
Permission denied: connect
Permission denied: connect
Permission denied: connect
Permission denied: connect
Brute-Force Attack Stopped!
```

```java
public Server(String rootDirectory, int port, WebServer wi
    this.rootDirectory = rootDirectory;
    // We obtain the file system of the Path
    this.port = port;
    this.stop = false;
```

192.168.1.114:8080/TestPlugin/Servletk

404 - We couldn't find the page you are looking for

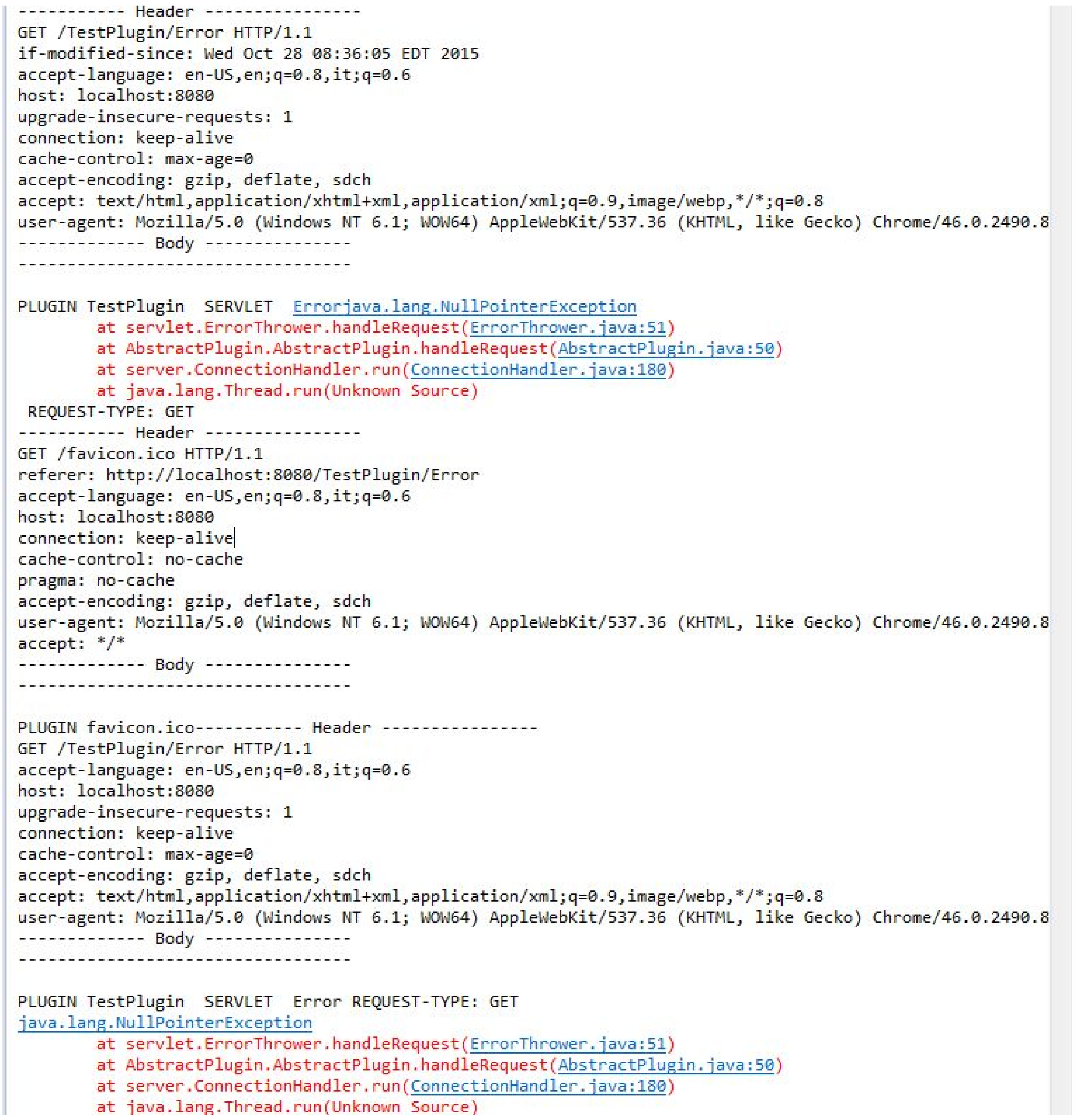
Name	Me...	Status	Type	Initiator	Size	Time
Servletk	GET	200	do...	Other	618 B	9 ms
jquery.min.js	GET	304	scr...	Servlet...	0 B	37 ms
5e7374300a84...	GET	304	jpeg	Servlet...	248 B	33 ms
favicon.ico	GET	400	tex...	Other	173 B	10 ms

b. Fault Prevention

i.

Source	Stimulus	Environment	Artifact	Response	Response Measure
Normal User	HTTP Request to a servlet with code that causes exception	Normal Conditions	HTTP Server with error laces servlet	The exception is caught the the server continues to work	The servlet is still available and so is the server

ii. Create a servlet code that causes an exception that Eclipse will not catch on compile, like accessing an array at index 99 that only has 1 item at index 0. This causes an error and we then check to make sure the server is still running, other servlets can be called, and the same servlet can be called again.



```
----------- Header ----------------
GET /TestPlugin/Error HTTP/1.1
if-modified-since: Wed Oct 28 08:36:05 EDT 2015
accept-language: en-US,en;q=0.8,it;q=0.6
host: localhost:8080
upgrade-insecure-requests: 1
connection: keep-alive
cache-control: max-age=0
accept-encoding: gzip, deflate, sdch
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
user-agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.8
------------- Body ---------------
----------------------------------

PLUGIN TestPlugin  SERVLET  Errorjava.lang.NullPointerException
        at servlet.ErrorThrower.handleRequest(ErrorThrower.java:51)
        at AbstractPlugin.AbstractPlugin.handleRequest(AbstractPlugin.java:50)
        at server.ConnectionHandler.run(ConnectionHandler.java:180)
        at java.lang.Thread.run(Unknown Source)
 REQUEST-TYPE: GET
----------- Header ----------------
GET /favicon.ico HTTP/1.1
referer: http://localhost:8080/TestPlugin/Error
accept-language: en-US,en;q=0.8,it;q=0.6
host: localhost:8080
connection: keep-alive
cache-control: no-cache
pragma: no-cache
accept-encoding: gzip, deflate, sdch
user-agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.8
accept: */*
------------- Body ---------------
----------------------------------

PLUGIN favicon.ico----------- Header ----------------
GET /TestPlugin/Error HTTP/1.1
accept-language: en-US,en;q=0.8,it;q=0.6
host: localhost:8080
upgrade-insecure-requests: 1
connection: keep-alive
cache-control: max-age=0
accept-encoding: gzip, deflate, sdch
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
user-agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.8
------------- Body ---------------
----------------------------------

PLUGIN TestPlugin  SERVLET  Error REQUEST-TYPE: GET
java.lang.NullPointerException
        at servlet.ErrorThrower.handleRequest(ErrorThrower.java:51)
        at AbstractPlugin.AbstractPlugin.handleRequest(AbstractPlugin.java:50)
        at server.ConnectionHandler.run(ConnectionHandler.java:180)
        at java.lang.Thread.run(Unknown Source)
```

iii. The servlet ran and returned the same response, not server error, so the number being not 404, or error that the server is unavailable.

iv. The improved tactic for the performance is the fault prevention in that there may be error in the code, but it does not break the server and cause a fault to the server. The individual servlet instead gets caught from its exception and proceeds to not break server, or itself.

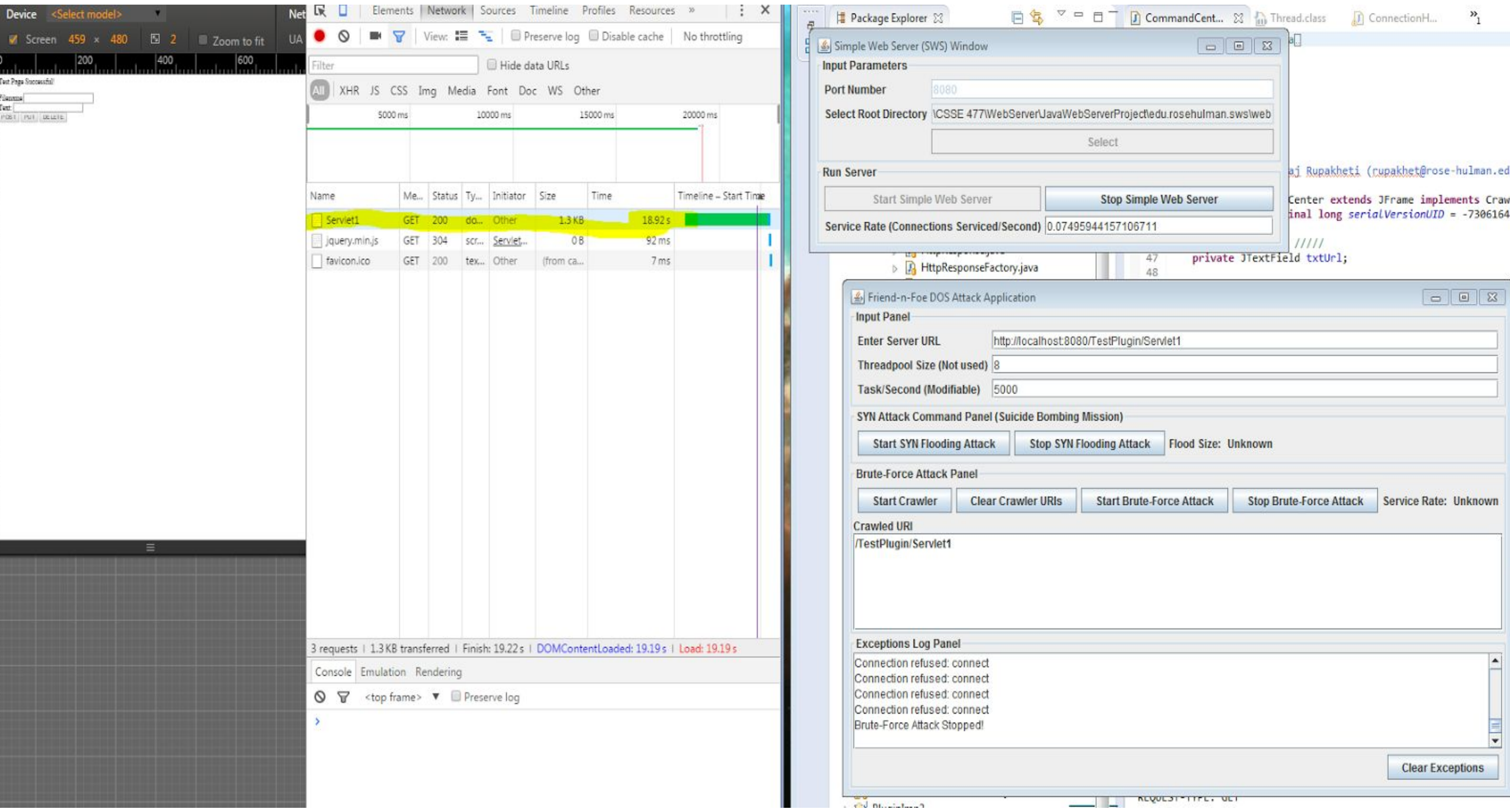
Comparing Tactics:

The implementation of DDoS prevention prevents malicious users from making the server unavailable, while the Fault Prevention prevents accidents, like faulty servlet code from breaking the system. One is from malicious user, the other one could be, but is not limited to that. Both work to keep the system running, DDoS from external, Fault Prevention from internal.

2. **Performance**
    a. DDoS Attack Prevention
        i.

Source	Stimulus	Environment	Artifact	Response	Response Measure
DDoS Server	Numerous HTTP Requests from same server	DDoS Attack	HTTP Server	DDoS is banned and normal users can make calls to server	User can make HTTP Request and receives a response within normal 2 seconds of before DDoS started

        ii. TTest the server before DDoS is being ran for a time until response. Then run the DDoS for a while and try to get a response within 5 seconds to know that the server is still available.
        iii. The image below shows a DDOS attack running against our unprotected server. With 5000 attacks per second, it takes the other client 18.92s to get a response from our server. We need to prevent this!



Name	Me...	Status	Ty...	Initiator	Size	Time
Servlet1	GET	200	do...	Other	1.3 KB	18.92 s
jquery.min.js	GET	304	scr...	Servlet...	0 B	92 ms
favicon.ico	GET	200	tex...	Other	(from ca...	7 ms

```
3 requests | 1.3 KB transferred | Finish: 19.22 s | DOMContentLoaded: 19.19 s | Load: 19.19 s
```

Simple Web Server (SWS) Window

Port Number: 8080
Select Root Directory: \CSSE 477\WebServer\JavaWebServerProject\edu.rosehulman.sws\web
Service Rate (Connections Serviced/Second): 0.07495944157106711

Friend-n-Foe DOS Attack Application

Enter Server URL: http://localhost:8080/TestPlugin/Servlet1
Threadpool Size (Not used): 8
Task/Second (Modifiable): 5000
Flood Size: Unknown
Service Rate: Unknown
Crawled URI: /TestPlugin/Servlet1

```
Connection refused: connect
Connection refused: connect
Connection refused: connect
Connection refused: connect
Brute-Force Attack Stopped!
```

iv. This was implemented by having a counter for each address. If the number of requests from that address surpasses our set limit, then the address is banned for a period of time. This is using a hashmap for the address, an incremented counter, and a limit. Then we also have reset clock that clears that hashmap after an interval. This prevents DDoS because we can track the IP and if it passes the certain number, we will be able to put the address on the ban list. With a DDoS attack, other users have to wait for the system, or they get completely locked out. This creates for a long time before their request may get handled, waiting for all the other DDoS requests.

b. Resource Control

i.

Source	Stimulus	Environment	Artifact	Response	Response Measure
Server	Perioud of Time	Normal Conditions	Timer Thread	After a set amount of time the tracking of addresses gets cleared	After 5 minutes, the tracking hashmap is cleared to empty.

ii. Have a user connect to the server and the server will print out the amount of times the address has connected. Connect multiple times and then wait 5 minutes. After 5 minutes the user will reconnect and the connection should return back to one.

iii. When the user connects to the server, after 5 minutes, the server should print only 1 recorded connection from that address.

iv. This tactic is for reducing resources in control resource demand in performance. If there were a lot of different addresses to connect, the memory of the hashmap which records the users would continue to expand and take up space. Having the hashmap resource be cleared periodically reduces memory resources that the server uses.

Comparing Tactics:

DDoS prevention works more in improving latency for a user that may be trying to send a request to the server during a DDoS attack. This takes up resources of the server, requiring a lot of the processing power to handle all of the requests, so reducing request from the DDoS prevention reduces strain on the server, while the resource control helps to keep the server controlled on memory as to prevent overuse in unnecessary amounts. This allows for the server to allocate memory elsewhere, especially if it is needed, where having the memory cleaned for the users connected will not be very important to users for speed and performance of the server.

3. **Security**

a. DDoS Attack Prevention

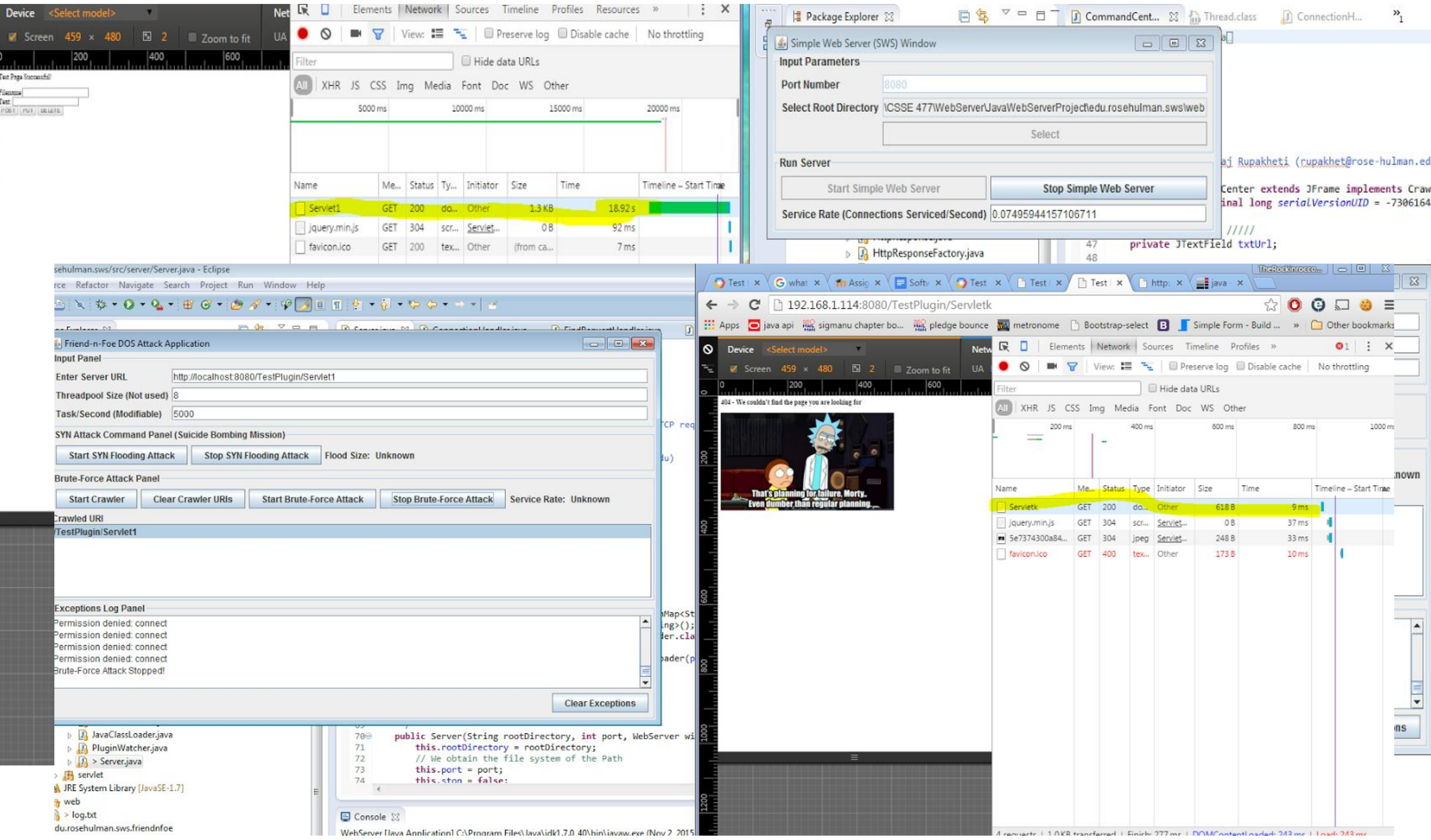
i.

Source	Stimulus	Environment	Artifact	Response	Response Measure
DDoS Server	Numerous HTTP Requests from same	DDoS Attack	HTTP Server	DDoS server added to	DDoS address is banned and cannot make any more

	server			ban list	requests

ii. .TTest the server before DDoS is being ran for a time until response. Then run the DDoS for a while and try to get a response within 5 seconds to know that the server is still available.

iii. The image below shows a DDOS attack running against our unprotected server. With 5000 attacks per second, it takes the other client 18.92s to get a response from our server. We need to prevent this!



Name	Me...	Status	Ty...	Initiator	Size	Time
Servlet1	GET	200	do...	Other	1.3 KB	18.92 s
jquery.min.js	GET	304	scr...	Servlet...	0 B	92 ms
favicon.ico	GET	200	tex...	Other	(from ca...	7 ms

Simple Web Server (SWS) Window

Input Parameters

Port Number: 8080

Select Root Directory: \CSSE 477\WebServer\JavaWebServerProject\edu.rosehulman.sws\web

Run Server

Service Rate (Connections Serviced/Second): 0.07495944157106711

Friend-n-Foe DOS Attack Application

Input Panel

Enter Server URL: http://localhost:8080/TestPlugin/Servlet1

Threadpool Size (Not used): 8

Task/Second (Modifiable): 5000

SYN Attack Command Panel (Suicide Bombing Mission)

Flood Size: Unknown

Brute-Force Attack Panel

Service Rate: Unknown

Crawled URI: /TestPlugin/Servlet1

Exceptions Log Panel

```
Permission denied: connect
Permission denied: connect
Permission denied: connect
Permission denied: connect
Brute-Force Attack Stopped!
```

```java
public Server(String rootDirectory, int port, WebServer w
    this.rootDirectory = rootDirectory;
    // We obtain the file system of the Path
    this.port = port;
    this.stop = false;
```

192.168.1.114:8080/TestPlugin/Servletk

Name	Me...	Status	Type	Initiator	Size	Time
Servletk	GET	200	do...	Other	618 B	9 ms
jquery.min.js	GET	304	scr...	Servlet...	0 B	37 ms
5e7374300a84...	GET	304	jpeg	Servlet...	248 B	33 ms
favicon.ico	GET	400	tex...	Other	173 B	10 ms

iv. This was implemented by having a counter for each address. If the number of requests from that address surpasses our set limit, then the address is banned for a period of time. This is using a hashmap for the address, an incremented counter, and a limit. Then we also have reset clock that clears that hashmap after an interval. This prevents DDoS because we can track the IP and if it passes the certain number, we will be able to put the address on the ban list. This works as a detective security measure to stop attacks beforehand from locking the system.

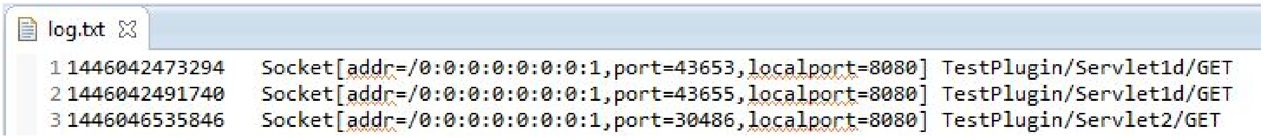
b. Logging System

i.

Source	Stimulus	Environment	Artifact	Response	Response Measure
Normal User	HTTP Request	Normal Conditions	HTTP Server	New socket user is logged	New line in Log File

ii. Previously there was no file, now we test that there is a file and the user data is recorded.

iii. There is 1 new line added to the log file that includes the user's information



```
1446042473294   Socket[addr=/0:0:0:0:0:0:0:1,port=43653,localport=8080] TestPlugin/Servlet1d/GET
1446042491740   Socket[addr=/0:0:0:0:0:0:0:1,port=43655,localport=8080] TestPlugin/Servlet1d/GET
1446046535846   Socket[addr=/0:0:0:0:0:0:0:1,port=30486,localport=8080] TestPlugin/Servlet2/GET
```

iv. This is not a preventative measure, but a recover tactic set in place to check if anything does seem amiss, to check the log files to see any malicious looking address, or at least having recorded the last addresses before the system may have gone wrong. This leaves a fingerprint to allow us to track down any malicious user so we are more secure in knowing what any bad attacks may be coming from. This was implemented by adding the request information into a log file for us to use.

Comparing Tactics:

The DDoS prevention is a resist security measure, while the logging is reactionary. DDoS prevention will help prevent the security issue right away while the logging will help afterwards in checking and reacting afterwards from any malicious attacks.

Future Improvements

1. Client Side Application to allow users to easily use and test the server.
2. Allowing of a user to send their priority on a HTTP request to allow a users to set the important that the system can be run.
3. Considering implementing a broker pattern if this allows for ease of use for the server, but requires a lot of work.