CMPE58E

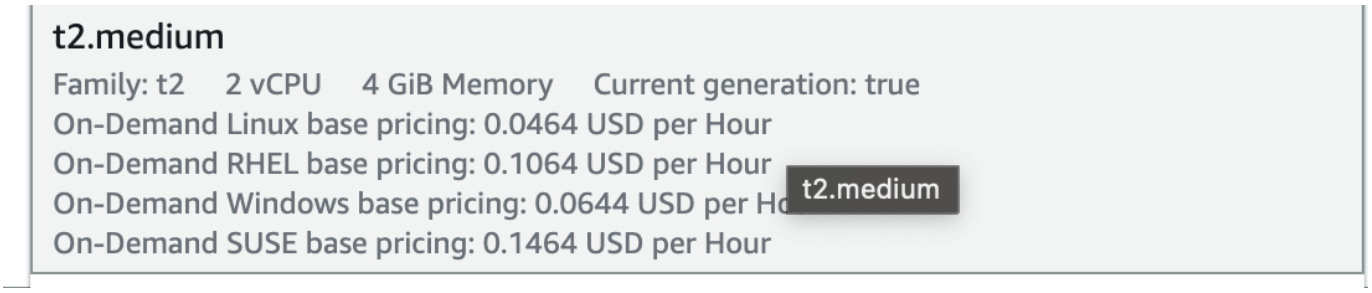
KADIR GOKHAN SEZER

Part I: Basic set-up

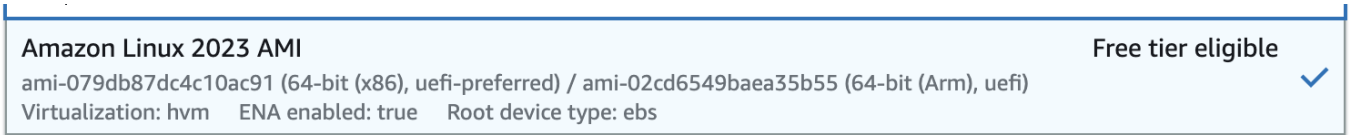
● Find a simple web server application. (Don’t write it yourself, find a project from github)
● Run your application on a public cloud service provider.
● Evaluate the performance of your system by sending synthetic data and measuring the response time experienced by the application users (Use a test suite such as locust or jmeter)
● Show your results through performance graphics

For this midterm, I have developed a straightforward web server that responds with the string 'pong' when you send a request to the **hostname/ping**. The server is hosted on AWS, and the machine type is t2.medium. I will employ Locust to assess the server's performance. In this midterm, I will present the parameters used in **Locust** and showcase the results.

The specifications of the EC2 instance that serves the web application are as follows:



and has also 16 GB storage. The OS of the EC2 is

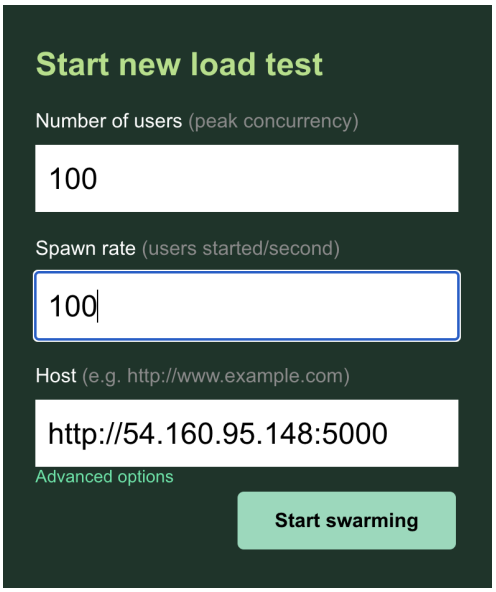


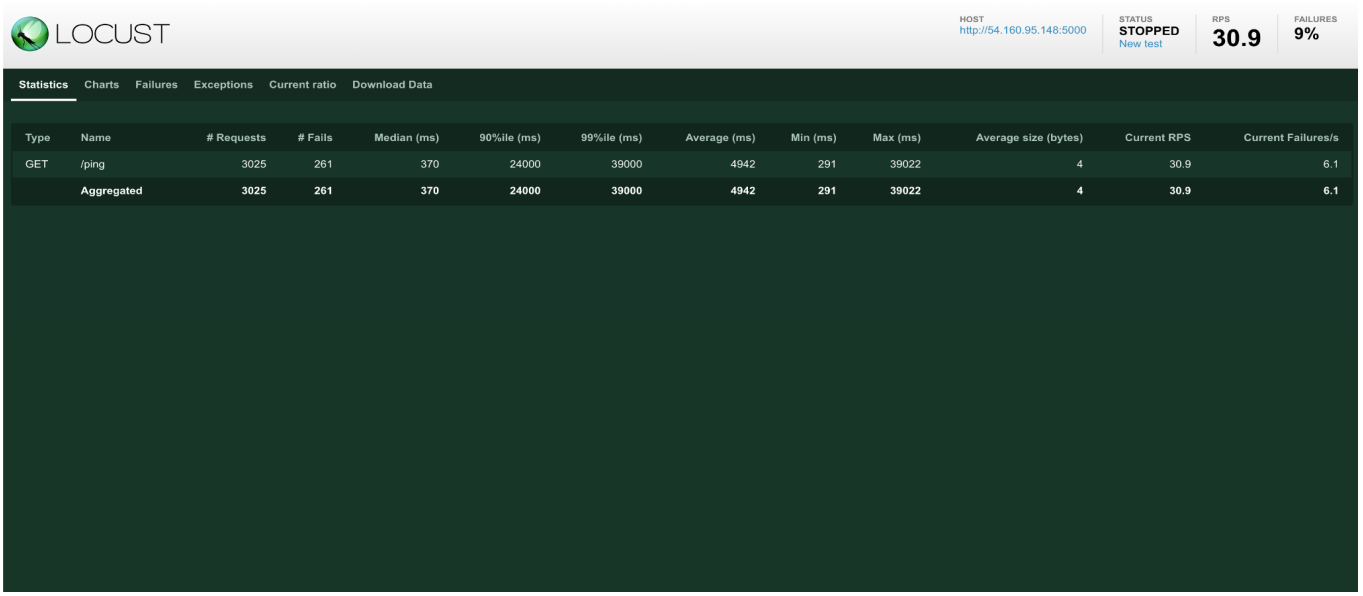
Versions of the programs on the EC2
Python 3.9.16
Flask 3.0.0
Werkzeug 3.0.1

Versions of the programs that are running on the client (the machine that performs the test**)**
Python 3.11.4
Flask 3.0.0
Werkzeug 3.0.1
locust 2.20.0 from /Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/locust (python 3.11.4)

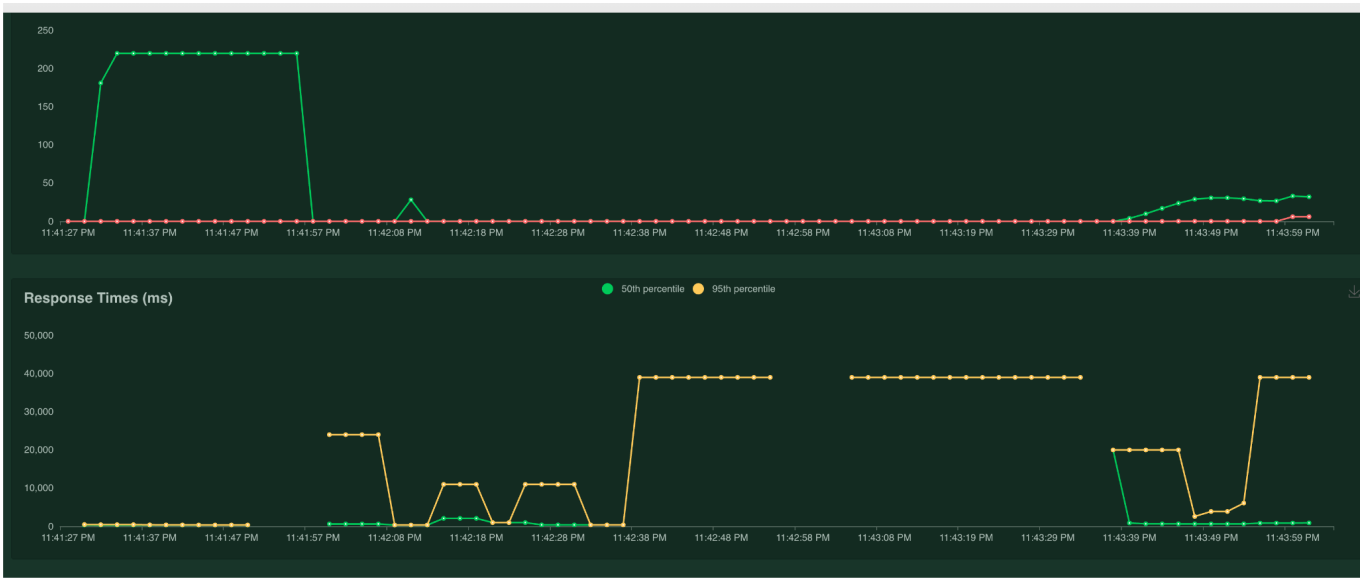
```python

from flask import Flask, make_response
from flask_cors import CORS, cross_origin
import sys

app = Flask(__name__)
cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'

@app.route('/ping', methods=['GET'])
@cross_origin()
def ping():
    print("pong", file=sys.stderr)
    resp = make_response("pong")
    return resp

if __name__ == '__main__':
    app.run(host="0.0.0.0", post=5000)

```

TEST 1

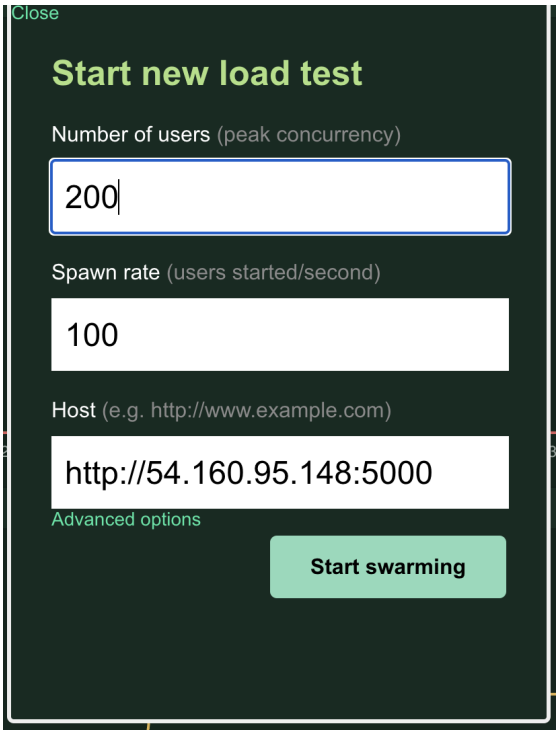




Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/ping	3025	261	370	24000	39000	4942	291	39022	4	30.9	6.1
	Aggregated	**3025**	**261**	**370**	**24000**	**39000**	**4942**	**291**	**39022**	**4**	**30.9**	**6.1**

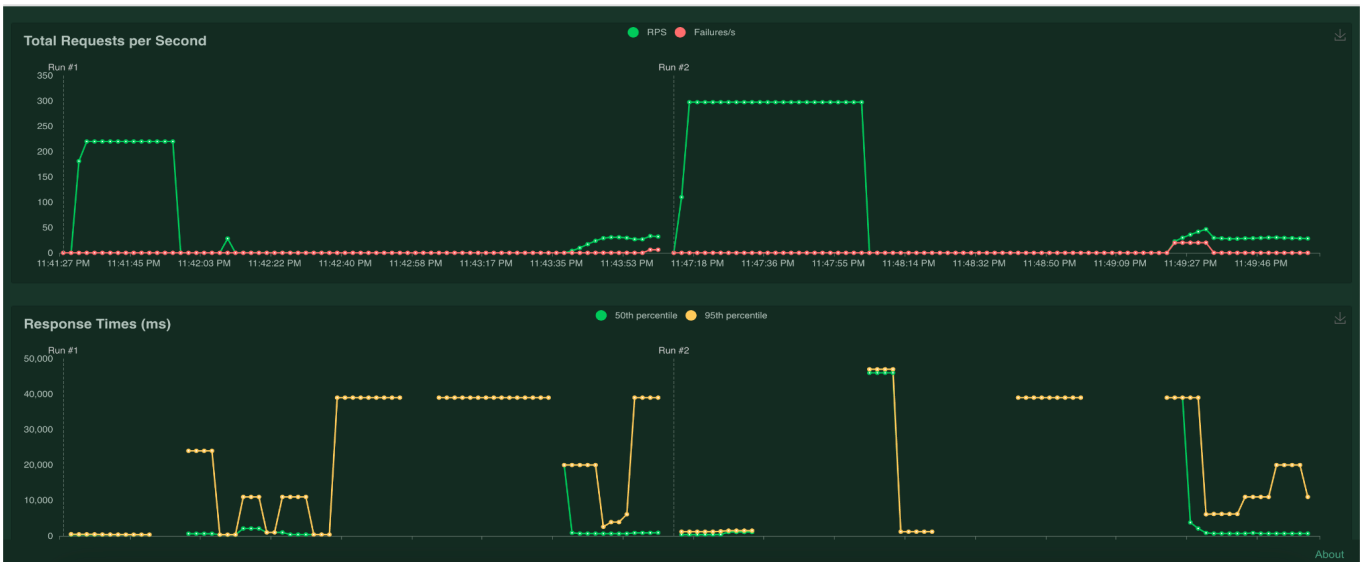


TEST 2

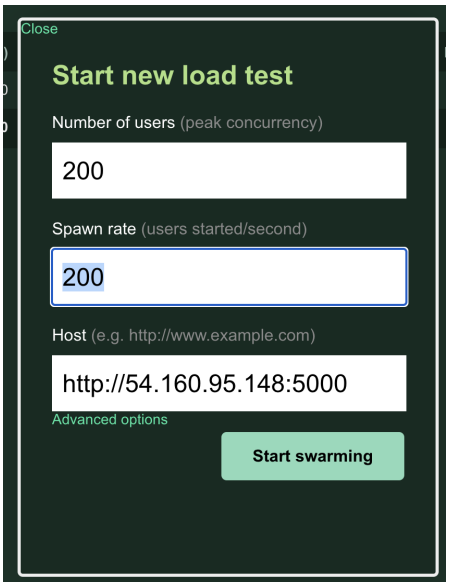


Statistics | Charts | Failures | Exceptions | Current ratio | Download Data

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/ping	3655	464	570	39000	47000	8344	290	47134	3	28.9	0
	Aggregated	**3655**	**464**	**570**	**39000**	**47000**	**8344**	**290**	**47134**	**3**	**28.9**	**0**



TEST 3



Statistics | Charts | Failures | Exceptions | Current ratio | Download Data

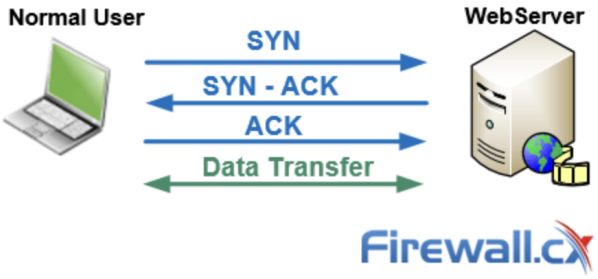
Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/ping	2539	494	790	39000	39000	9065	291	39015	3	13.5	13.5
	Aggregated	**2539**	**494**	**790**	**39000**	**39000**	**9065**	**291**	**39015**	**3**	**13.5**	**13.5**

Part II: The attack

● Simulate a simple DDOS attack (TCP SYN) to your application. Make sure you can play with the severity of the attack through your configuration parameters.
● Carry-out systematic tests to show how the performance of the system you reported on Part I is altered.
● Show your results via performance graphics.
● Discuss how the performance is changing as the attack becomes more severe.

Firstly, I searched for a way to perform this TCP SYN DDOS attack.
https://www.firewall.cx/tools-tips-reviews/network-protocol-analyzers/performing-tcp-syn-flood-attack-and-detecting-it-with-wireshark.html

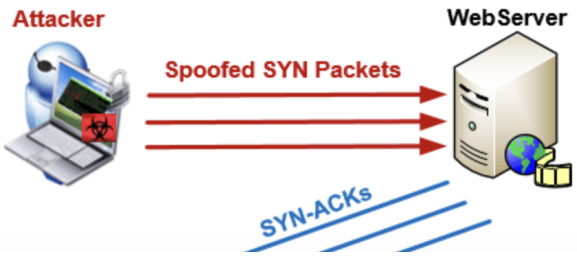
When a client attempts to connect to a server using the **TCP protocol** e.g (HTTP or HTTPS), it is first required to perform a **three-way handshake** before any data is exchanged between the two. Since the **three-way TCP handshake** is always initiated by the client it sends a **SYN packet** to the **server**.



The server next replies acknowledging the request and at the same time sends its own **SYN request** – this is the **SYN-ACK packet**. The finally the client sends an **ACK packet** which confirms both two hosts agree to create a connection. The connection is therefore established and data can be transferred between them.

Read our TCP Overview article for more information on the 3-way handshake

In a **SYN flood**, the attacker sends a **high volume of SYN packets** to the server using **spoofed IP addresses** causing the server to send a reply (SYN-ACK) and leave its ports half-open, awaiting for a reply from a host that doesn't exist:



This link helped me how to attack. Then, I initiated a EC2 on AWS again, whose OS is Kali Linux.
The specifications of the EC2 that attacks

Amazon Machine Image (AMI)

kali-last-snapshot-amd64-2023.4.0-804fcc46-63fc-4eb6-85a1-50e66d6c7215
ami-02d46314883bdd49c

Verified provider

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Catalog	Published	Architecture	Virtualization	Root device type	ENA Enabled
AWS Marketplace AMIs	2023-12-05T17:32:52.000Z	x86_64	hvm	ebs	Yes

Instance type

t2.medium
Family: t2 2 vCPU 4 GiB Memory Current generation: true

Since the website suggests me to use hping3, I learnt what it does and how it does that. This website helped me to understand. Then, I performed an attack with these command.

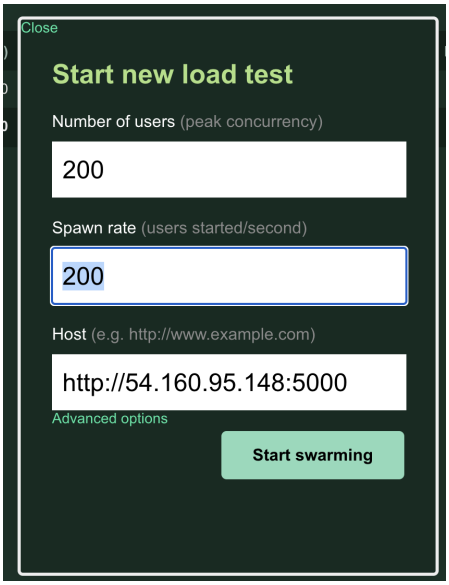
https://www.kali.org/tools/hping3/
The flags are being mentioned in this text:

"We're sending 15000 packets (-c 15000) at a size of 120 bytes (-d 120) each. We're specifying that the SYN Flag (-S) should be enabled, with a TCP window size of 64 (-w 64). To direct the attack to our victum's HTTP web server we specify port 80 (-p 80) and use the --flood flag to send packets as fast as possible. As you'd expect, the --rand-source flag generates spoofed IP addresses to disguise the real source and avoid detection but at the same time stop the victim's SYN-ACK reply packets from reaching the attacker."

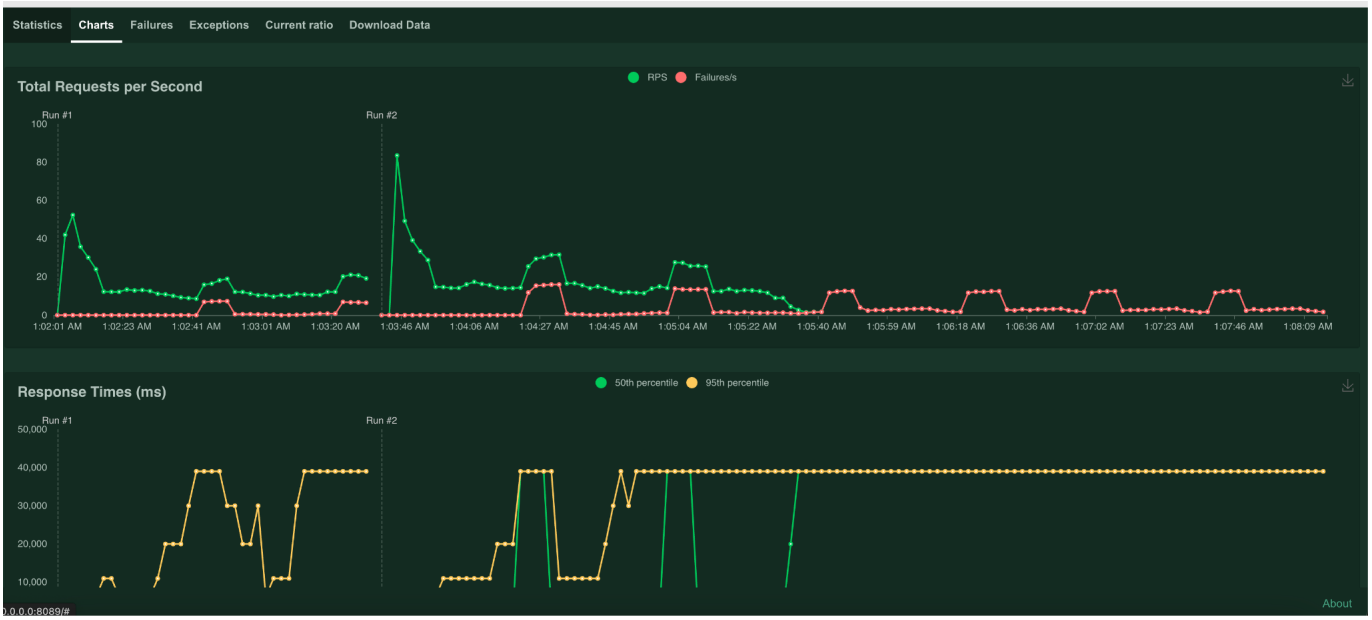
```
usage: hping3 host [options]
  -c  --count      packet count
  -d  --data       data size
  -S  --syn        set SYN flag
  -w  --win        winsize (default 64)
  -p  --destport   [+][+]<port> destination port(default 0) ctrl+z inc/dec
      --flood    sent packets as fast as possible. Don't show replies.
      --rand-source    random source address mode. see the man.
```

```
┌──(kali㉿kali)-[~]
└─$ sudo hping3 -c 15000 -d 120 -S -w 64 -p 5000 --flood --rand-source 54.160.95.148
HPING 54.160.95.148 (eth0 54.160.95.148): S set, 40 headers + 120 data bytes
hping in flood mode, no replies will be shown
```

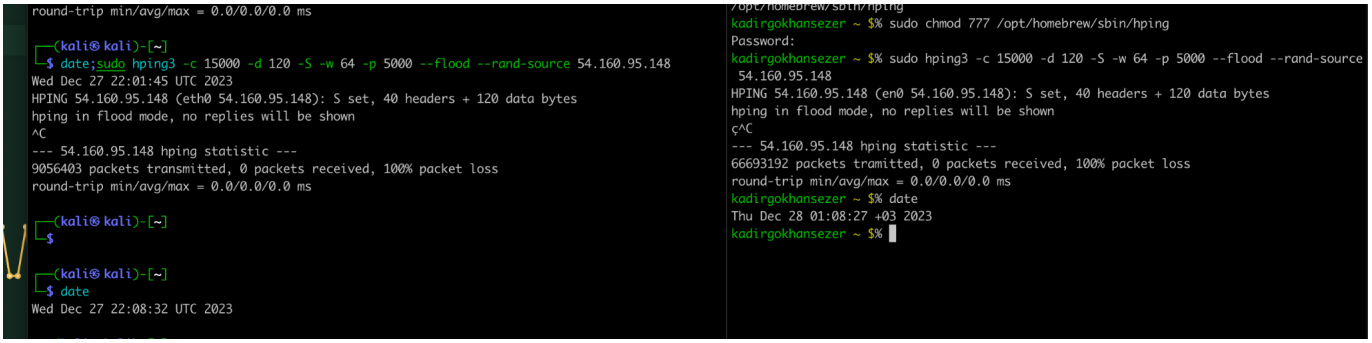
Then I performed the same tests again.

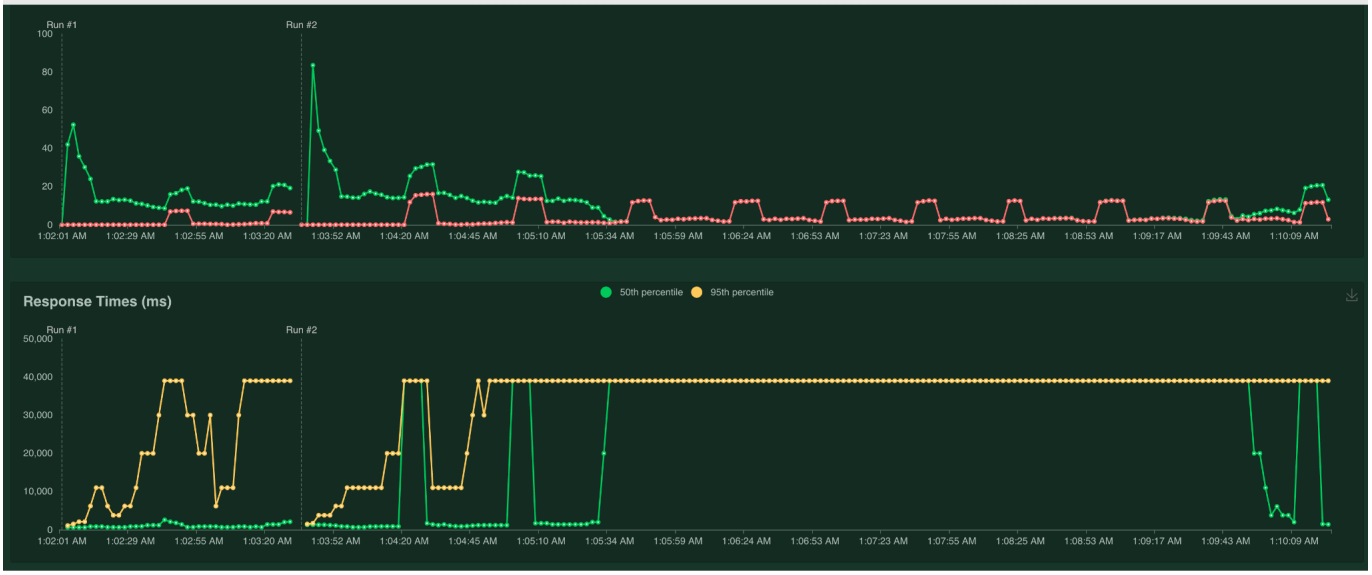


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	2676	1136	3800	39000	39000	17906	291	39028	2	3.2	3.2
	Aggregated	**2676**	**1136**	**3800**	**39000**	**39000**	**17906**	**291**	**39028**	**2**	**3.2**	**3.2**



After that, I stopped the DDoS attack to observe when the system stabilizes. With the 'date' command in the two terminals (that are attacking together), you can observe the time it takes to stop. Additionally, you may notice that the time it takes to recover is quite long.

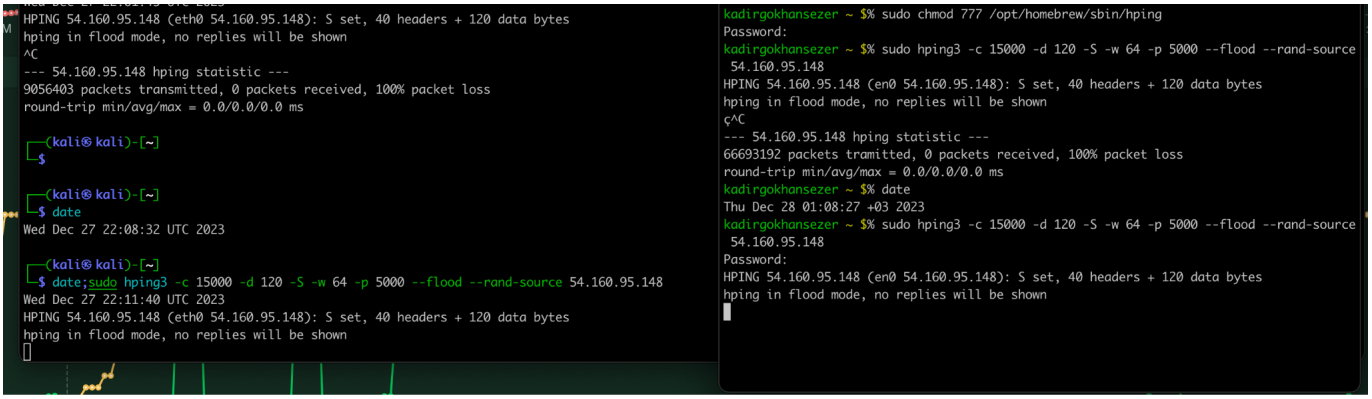


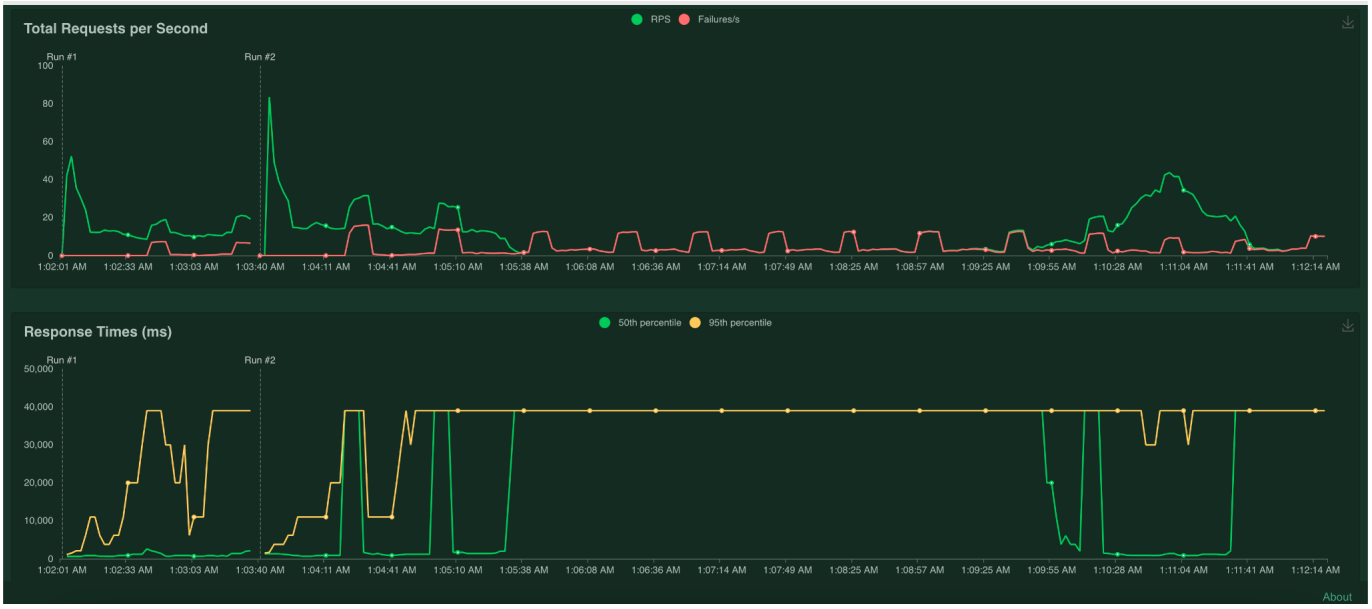


Statistics | Charts | Failures | Exceptions | Current ratio | Download Data

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	4227	1910	6200	39000	39000	19035	291	45862	2	32	2.4
	Aggregated	**4227**	**1910**	**6200**	**39000**	**39000**	**19035**	**291**	**45862**	**2**	**32**	**2.4**

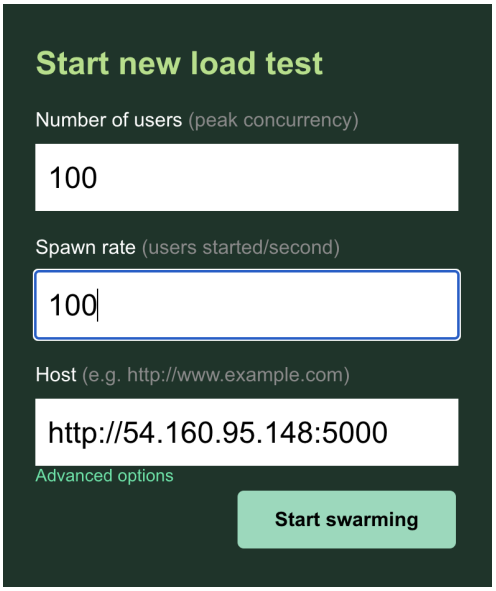
After that, I started the attack again.

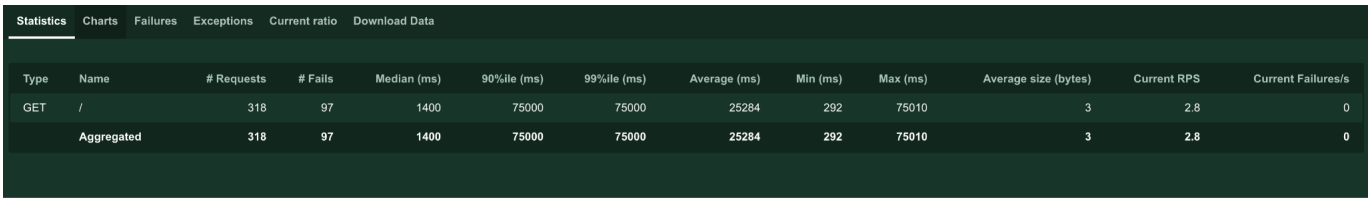




Then, you see it was stuck again. It can really impact the experience of the users. It makes me feel afraid; anyone can have this power?

Another test with new parameters.



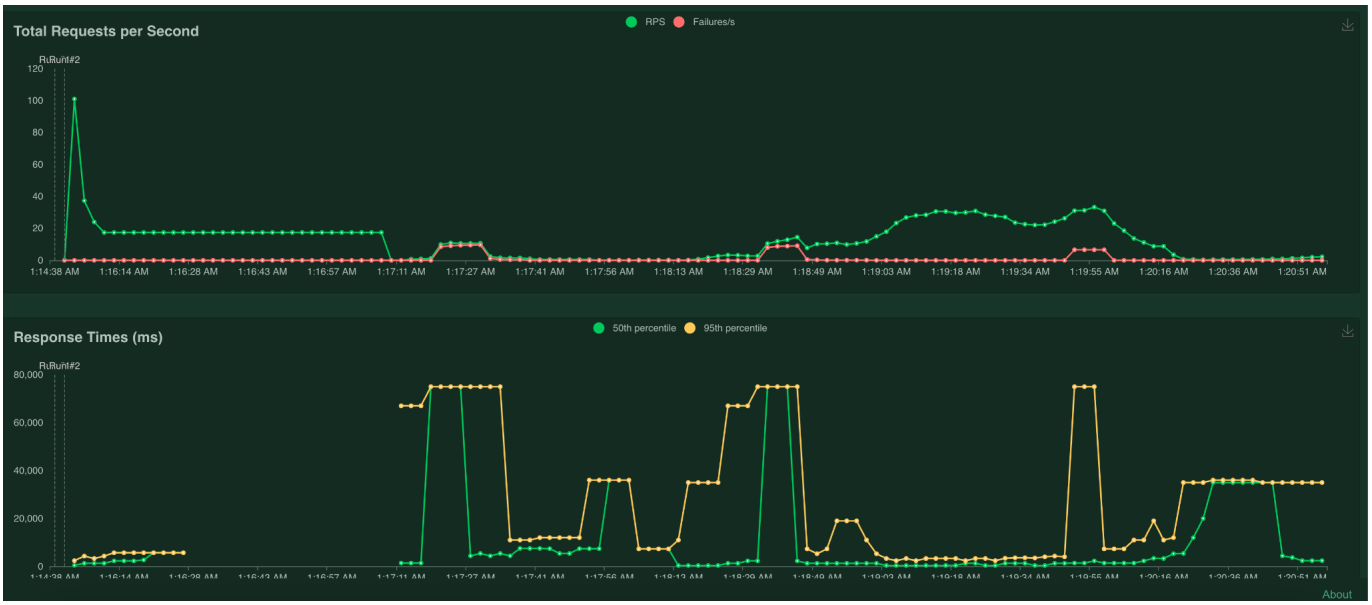


Statistics | Charts | Failures | Exceptions | Current ratio | Download Data

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	318	97	1400	75000	75000	25284	292	75010	3	2.8	0
	Aggregated	**318**	**97**	**1400**	**75000**	**75000**	**25284**	**292**	**75010**	**3**	**2.8**	**0**

You see, in this case, I completely blocked the system. Any request received no response during this time.

After I stopped the attack, it started to recover.



This attempt is just showing the way I played with the DDoS power. You can see the response time..

Results:

I can say that we were remotely aware of this topic, which people frequently discuss. Theoretically, we were knowledgeable about the subject. However, getting our hands dirty was quite enlightening. I was able to have a general command of the topic. Looking at the results, I can directly influence the normally displayed performance with DDoS attacks. I make the decision entirely. This is both intimidating and exciting.

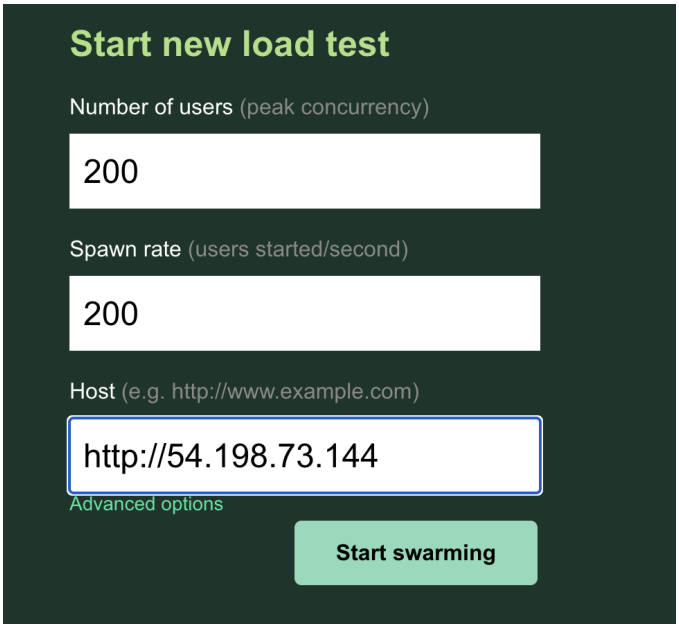
Also, I must mention that I could not conduct these tests in a completely objective environment. While performing these experiments, my internet was blocked several times. This could have been done by the ISP, or Amazon might have blocked my requests from outside; I cannot be sure. Speaking of Amazon, we know that they have teams set up to prevent DDoS attacks from EC2 instances and additional teams to prevent DDoS attacks on themselves. Therefore, I could manipulate these values, but I might have caught their attention. In the end, everything went smoothly. I learned that I could alter the response time with DDoS attacks.

Bonus1: If you apply a method to mitigate the attack and be able to show that your work enhances the performance you will get a 10p bonus. Your mitigation should not necessarily lower the attack impact down to zero, any significant improvement is acceptable.

While I was looking for a way to mitigate SYN flood attack, I found this website.
https://iserversupport.com/blog/how-to-block-or-prevent-syn-floods-attack/

And it says these will help the server to mitigate the attack.

echo 1 > /proc/sys/net/ipv4/tcp_syncookies
echo 2048 > /proc/sys/net/ipv4/tcp_max_syn_backlog
echo 3 > /proc/sys/net/ipv4/tcp_synack_retries



```
┌──(kali㉿kali)-[~]
└─$ date;sudo hping3 -c 15000 -d 120 -S -w 64 -p 5000 --flood --rand-source 54.198.73.144
Thu Dec 28 19:08:19 UTC 2023
HPING 54.198.73.144 (eth0 54.198.73.144): S set, 40 headers + 120 data bytes
hping in flood mode, no replies will be shown
```

Statistics | Charts | Failures | Exceptions | Current ratio | Download Data

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	2091	892	2900	25000	25000	11897	323	25523	2	16	15.6
	Aggregated	**2091**	**892**	**2900**	**25000**	**25000**	**11897**	**323**	**25523**	**2**	**16**	**15.6**

From the results you can see that Average response time is better now.
It was 17900, now 11897 which means the response time is %40 better.

Bonus2: If you can detect that you are under attack (without any information flowing to your detection system from the simulation setup), you will get a 10p bonus.

For detecting the DDoS attack
https://medium.com/@digestacademy/what-is-a-syn-flood-attack-ddos-how-to-detect-them-515ad4d7e1ee
https://www.geeksforgeeks.org/how-to-install-and-use-wireshark-on-ubuntu-linux/

These websites suggest using WireShark, which I already use. As you can see in the screenshot, all these packets are just coming to the server, and they are only SYN packets. With this way, we can detect the DDoS SYN attacks.