CYB102 Project 4

(🔗 Instructions Page)

👩🏿 Student Name: Didier Joseph DESMANGLES
✉️ Student Email: djdesmangles@gmail.com

Reflection (Required)

🤔 **Reflection Question #1:** If I had to **explain "what is a proxy server" in 3 emojis,** they would be…
(Feel free to put other comments about your experience in this unit here, too!)

💻↔️🌐
A proxy server ↔️ is an intermediary system that sits between a client 💻 (like a web browser) and a destination server 🌐(such as a website). It handles requests from the client, forwards them to the target server, and then sends the server's response back to the client.
💻 Client
↔️ Proxy handling requests between client and Internet
🌐 Internet

🧠**Reflection Question #2:** What are some different types of DoS/DDoS attacks?

1. **Volume-Based Attacks**
These attacks aim to overwhelm the target's bandwidth with a high traffic volume.
- *UDP Flood*: Sends a large number of User Datagram Protocol (UDP) packets to random ports on the target, causing the server to check for applications listening on those ports and respond with ICMP "Destination Unreachable" packets.
- *ICMP Flood*: Sends a massive amount of Internet Control Message Protocol (ICMP) Echo Request (ping) packets to the target, overwhelming its ability to respond to legitimate traffic.
- *TCP SYN Flood*: Exploits the TCP handshake process by sending a flood of SYN requests while never completing the handshake, causing the server to exhaust its resources waiting for the final ACK.

2. **Protocol Attacks**
These attacks exploit weaknesses in network protocols to consume server resources or network equipment.
- *Ping of Death*: Sends malformed or oversized packets to the target, causing buffer overflows and crashes.
- *Smurf Attack*: Uses ICMP packets directed to a broadcast address to amplify traffic,

causing a flood of responses to the victim.

- *SYN-ACK Flood*: Similar to SYN Floods, this attack sends a large number of SYN-ACK packets, overwhelming the target's ability to process them.

3. **Application Layer Attacks**
These attacks target specific applications or services, aiming to disrupt their functionality.

- *HTTP Flood*: Sends a flood of HTTP requests to a web server, overwhelming its resources and causing slowdowns or crashes.
- ***Slowloris***: Maintains many connections to the target web server while sending partial HTTP requests, causing the server to exhaust its connection pool.
- *DNS Query Flood*: Sends a high volume of DNS requests to a DNS server, overwhelming its resources and potentially causing service disruptions.

4. **Resource Exhaustion Attacks**
These attacks specifically aim to consume resources on the server.

- *DNS Amplification*: Spoofs the victim's IP address in a DNS query, causing DNS servers to send large responses to the victim.
- *Connection Exhaustion*: Attempts to open as many connections as possible to a service, exhausting the available connections and preventing legitimate users from connecting.

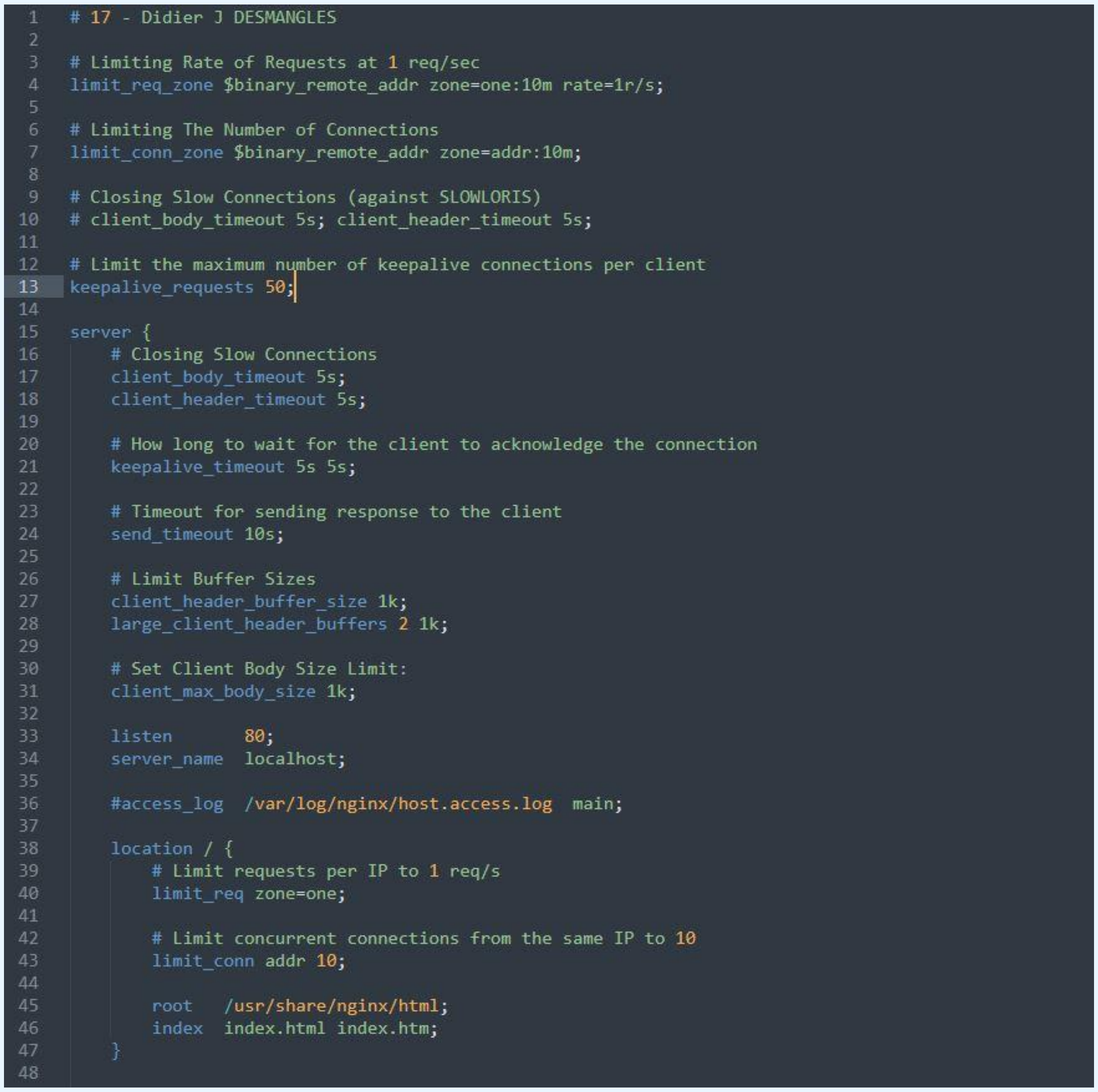
5. **Multi-Vector Attacks**
These attacks combine different techniques to overwhelm the target using various attack vectors simultaneously, making them harder to defend against.

📣 **Shoutouts:** Share appreciation for anyone who helped you out with this project or made your day a little better!

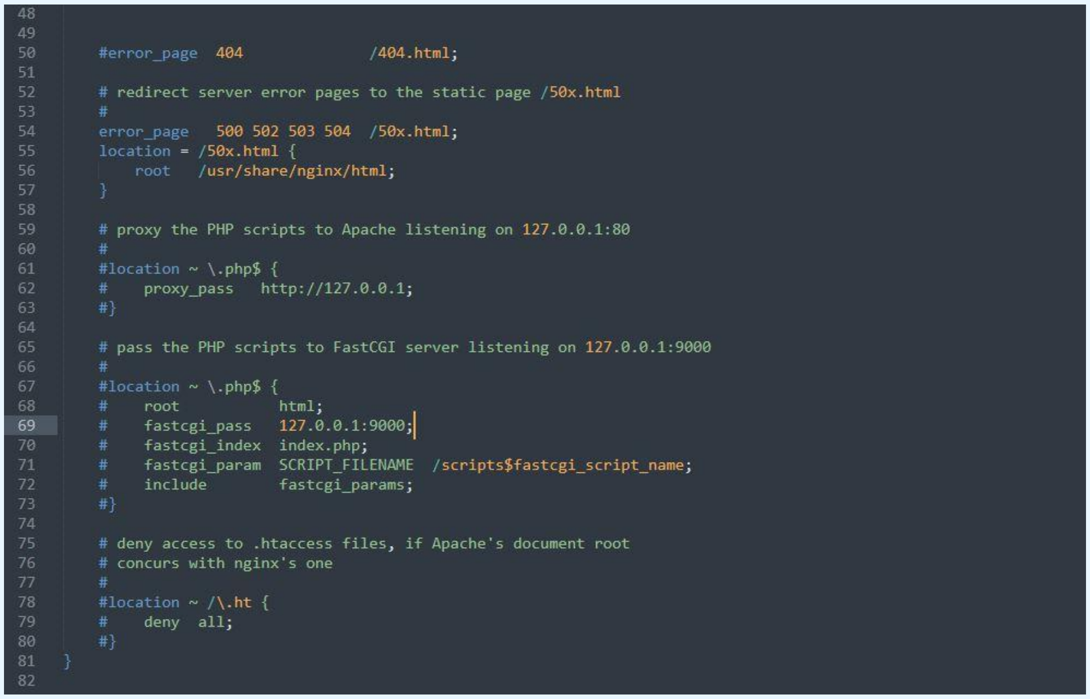
All my pals from TEAM 17 !! their questions force us to improve !! 😀

Required Challenges (Required)

Item #1: A screenshot of your `/etc/nginx/conf.d/default.conf` file with your DoS mitigation rules implemented:



```
# 17 - Didier J DESMANGLES

# Limiting Rate of Requests at 1 req/sec
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

# Limiting The Number of Connections
limit_conn_zone $binary_remote_addr zone=addr:10m;

# Closing Slow Connections (against SLOWLORIS)
# client_body_timeout 5s; client_header_timeout 5s;

# Limit the maximum number of keepalive connections per client
keepalive_requests 50;

server {
    # Closing Slow Connections
    client_body_timeout 5s;
    client_header_timeout 5s;

    # How long to wait for the client to acknowledge the connection
    keepalive_timeout 5s 5s;

    # Timeout for sending response to the client
    send_timeout 10s;

    # Limit Buffer Sizes
    client_header_buffer_size 1k;
    large_client_header_buffers 2 1k;

    # Set Client Body Size Limit:
    client_max_body_size 1k;

    listen       80;
    server_name  localhost;

    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        # Limit requests per IP to 1 req/s
        limit_req zone=one;

        # Limit concurrent connections from the same IP to 10
        limit_conn addr 10;

        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

```



```


    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
}

```

Note (Optional):
Slowloris is a type of application layer attack, specifically designed to exhaust the resources of a web server. Here's how it works:

Characteristics of Slowloris

- **Partial Requests**: Slowloris sends a series of incomplete HTTP requests to the target server, holding connections open by sending partial header information. It never completes the request, which keeps the connection active.
- **Resource Exhaustion**: By maintaining numerous open connections without completing them, Slowloris can exhaust the server's connection pool. Most web servers have a limit on the number of simultaneous connections they can handle, and Slowloris takes advantage of this by tying up those connections.
- **Low Bandwidth**: One of the distinguishing features of Slowloris is that it doesn't require a large amount of bandwidth. It can operate effectively with just a small amount of traffic, making it relatively stealthy compared to traditional volume-based attacks.
- **Target Specificity**: Slowloris is particularly effective against servers that are not well-configured to handle slow connections or that have lengthy timeout periods for requests.

Attack Mechanism

- **Connection Establishment**: Slowloris initiates connections to the target web server, sending a valid HTTP header but never completing the request.
- **Keep-Alive:** It uses the HTTP keep-alive feature to keep connections open without sending any further data, thereby preventing the server from closing those connections.
- **Exploitation**: As Slowloris keeps sending partial requests, it can eventually exhaust the server's available connections, leading to service unavailability for legitimate users.

Mitigation Strategies
To defend against Slowloris attacks, you can implement several strategies:

- *Reduce Timeout Values*: Lowering the timeouts for client connections can help close idle connections more quickly.
- *Limit Connection Rates*: Use connection rate limiting to prevent any single IP from opening too many simultaneous connections.
- *Limiting the Rate of Requests*: Limiting the rate at which the server accepts incoming requests to a value typical for real users.
- *Closing Slow Connections*: You can close connections that are writing data too infrequently, which can represent an attempt to keep connections open as long as possible (thus reducing the server's ability to accept new connections). Slowloris is an example of this type of attack. The *client_body_timeout* directive controls how long NGINX waits between writes of the client body, and the *client_header_timeout* directive controls how long NGINX waits between writes of client headers. The default for both directives is 60 seconds.

(Other techniques)

- *Denying IP Addresses*
- *Using Caching to Smooth Traffic Spikes*:
- *Blocking Requests*:
- *Increase Connection Limits/Handling High Loads*. Temporary fix

Our implementation:

```
# Limiting Rate of Requests at 1 req/sec.
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

# Limiting The Number of Connections
limit_conn_zone $binary_remote_addr zone=addr:10m;

# Closing Slow Connections against SLOWLORIS
client_body_timeout 5s; client_header_timeout 5s;

# Limit the maximum number of keepalive connections per client
keepalive_requests 50;

 # How long to wait for the client to acknowledge the connection
keepalive_timeout 5s 5s;

# Timeout for sending a response to the client
send_timeout 10s;

# Limit Buffer Sizes
client_header_buffer_size 1k;
large_client_header_buffers 2 1k;

# Set Client Body Size Limit:
client_max_body_size 1k;
```

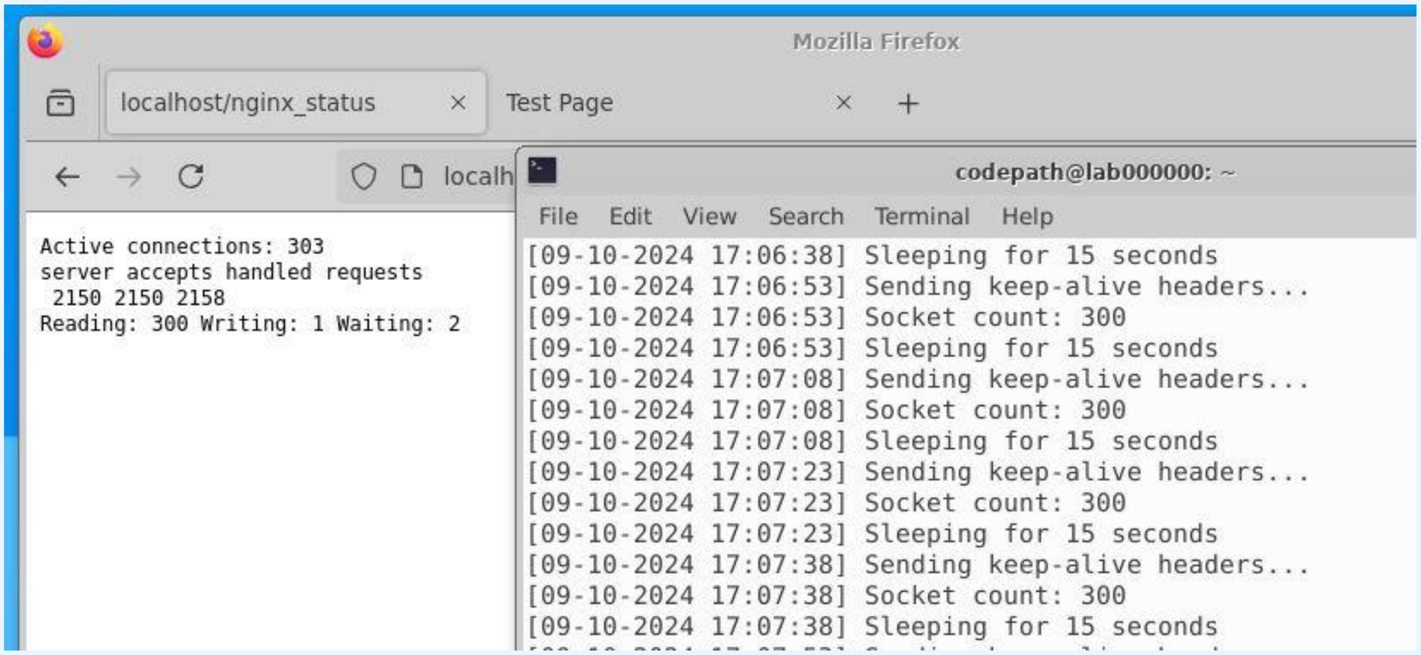
Item #2: A detailed explanation (two sentences minimum) of how you know that your DoS mitigation rules are working:

We execute slowloris : `slowloris -s 300 -v 127.0.0.1`

The Limiting Rate should be respected

```
# Limiting Rate of Requests at 1 req/sec.
limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
```

The nginx_status module is showing :



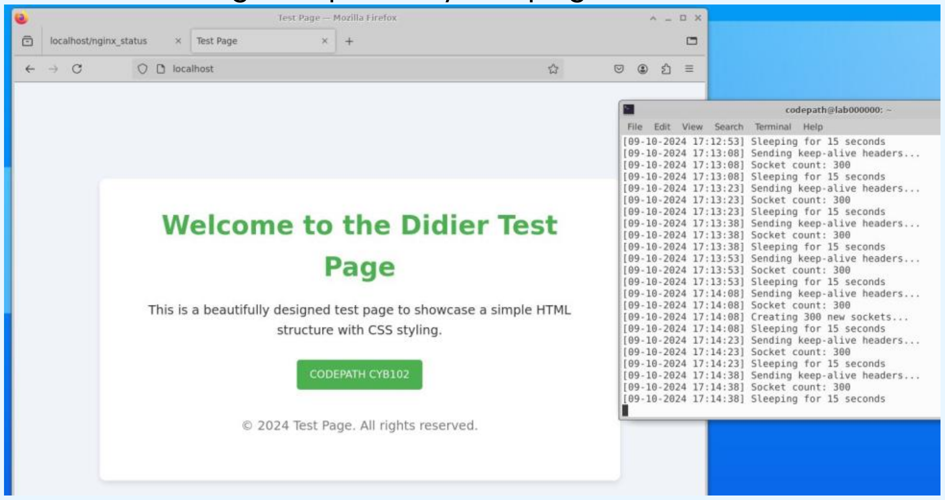
```
Active connections: 303
server accepts handled requests
 2150 2150 2158
Reading: 300 Writing: 1 Waiting: 2
```

```
[09-10-2024 17:06:38] Sleeping for 15 seconds
[09-10-2024 17:06:53] Sending keep-alive headers...
[09-10-2024 17:06:53] Socket count: 300
[09-10-2024 17:06:53] Sleeping for 15 seconds
[09-10-2024 17:07:08] Sending keep-alive headers...
[09-10-2024 17:07:08] Socket count: 300
[09-10-2024 17:07:08] Sleeping for 15 seconds
[09-10-2024 17:07:23] Sending keep-alive headers...
[09-10-2024 17:07:23] Socket count: 300
[09-10-2024 17:07:23] Sleeping for 15 seconds
[09-10-2024 17:07:38] Sending keep-alive headers...
[09-10-2024 17:07:38] Socket count: 300
[09-10-2024 17:07:38] Sleeping for 15 seconds
```

303 = Number of all open connections,
2150 = Accepted connections,
2150 = Handled connections
2158 = Handles requests

Requests per connection = handles requests / handled connections
Requests per connection = 2158 / 2150 = **1.0037**
This number fits the "Limiting Rate of Requests at 1 req/sec".

The Server should continue running while it is under attack.
By opening a browser and hitting F5 repeatedly, the page should still be accessible smoothly.



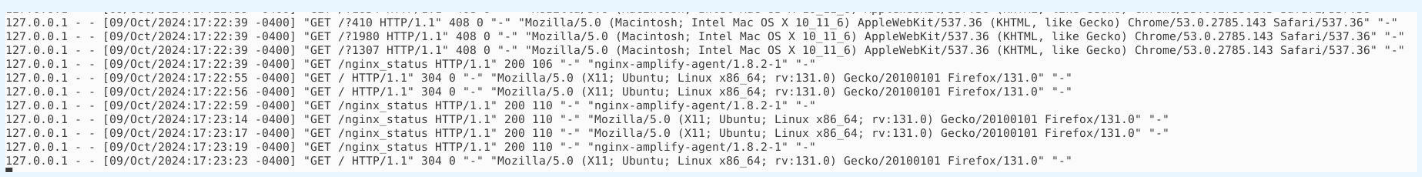
Welcome to the Didier Test Page

This is a beautifully designed test page to showcase a simple HTML structure with CSS styling.

© 2024 Test Page. All rights reserved.

```
[09-10-2024 17:12:53] Sleeping for 15 seconds
[09-10-2024 17:13:08] Sending keep-alive headers...
[09-10-2024 17:13:08] Socket count: 300
[09-10-2024 17:13:08] Sleeping for 15 seconds
[09-10-2024 17:13:23] Sending keep-alive headers...
[09-10-2024 17:13:23] Socket count: 300
[09-10-2024 17:13:23] Sleeping for 15 seconds
[09-10-2024 17:13:38] Sending keep-alive headers...
[09-10-2024 17:13:38] Socket count: 300
[09-10-2024 17:13:38] Sleeping for 15 seconds
[09-10-2024 17:13:53] Sending keep-alive headers...
[09-10-2024 17:13:53] Socket count: 300
[09-10-2024 17:13:53] Sleeping for 15 seconds
[09-10-2024 17:14:08] Sending keep-alive headers...
[09-10-2024 17:14:08] Socket count: 300
[09-10-2024 17:14:08] Creating 300 new sockets...
[09-10-2024 17:14:08] Sleeping for 15 seconds
[09-10-2024 17:14:23] Sending keep-alive headers...
[09-10-2024 17:14:23] Socket count: 300
[09-10-2024 17:14:23] Sleeping for 15 seconds
[09-10-2024 17:14:38] Sending keep-alive headers...
[09-10-2024 17:14:38] Socket count: 300
[09-10-2024 17:14:38] Sleeping for 15 seconds
```

Access Log
"sudo tail -f /var/log/nginx/access.log" produces the output below. The "AppleWebKIt" agent is from Slowloris. The "Mozilla" agent is from firefox, accessing http://localhost/nginx_status and http://localhost/index.html (Didier Test Page)



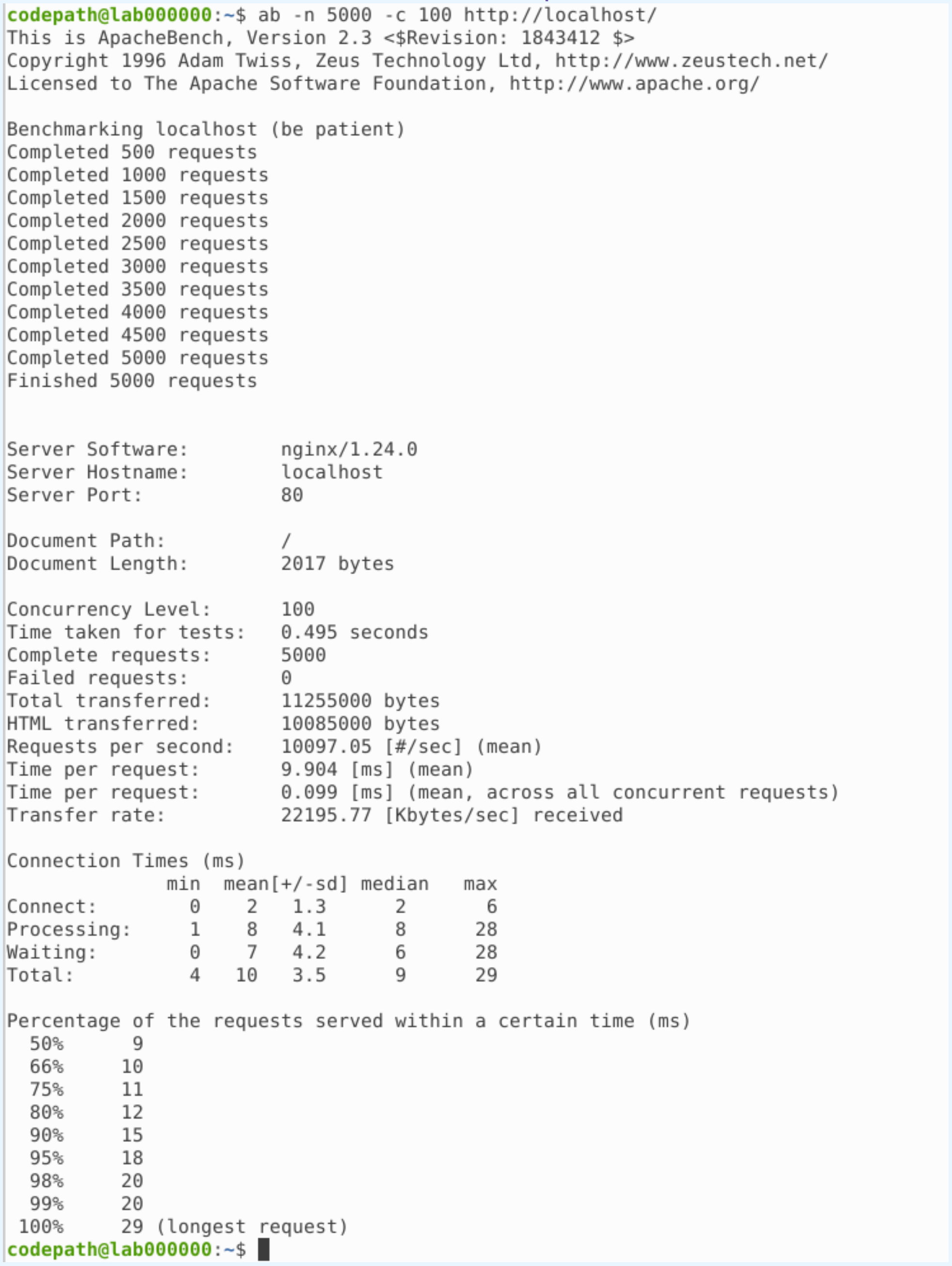
```
127.0.0.1 - - [09/Oct/2024:17:22:39 -0400] "GET /?410 HTTP/1.1" 408 0 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36" "-"
127.0.0.1 - - [09/Oct/2024:17:22:39 -0400] "GET /?1980 HTTP/1.1" 408 0 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36" "-"
127.0.0.1 - - [09/Oct/2024:17:22:39 -0400] "GET /?1307 HTTP/1.1" 408 0 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36" "-"
127.0.0.1 - - [09/Oct/2024:17:22:39 -0400] "GET /nginx_status HTTP/1.1" 200 106 "-" "nginx-amplify-agent/1.8.2-1" "-"
127.0.0.1 - - [09/Oct/2024:17:22:55 -0400] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0" "-"
127.0.0.1 - - [09/Oct/2024:17:22:56 -0400] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0" "-"
127.0.0.1 - - [09/Oct/2024:17:22:59 -0400] "GET /nginx_status HTTP/1.1" 200 110 "-" "nginx-amplify-agent/1.8.2-1" "-"
127.0.0.1 - - [09/Oct/2024:17:23:14 -0400] "GET /nginx_status HTTP/1.1" 200 110 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0" "-"
127.0.0.1 - - [09/Oct/2024:17:23:17 -0400] "GET /nginx_status HTTP/1.1" 200 110 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0" "-"
127.0.0.1 - - [09/Oct/2024:17:23:19 -0400] "GET /nginx_status HTTP/1.1" 200 110 "-" "nginx-amplify-agent/1.8.2-1" "-"
127.0.0.1 - - [09/Oct/2024:17:23:23 -0400] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0" "-"
```

Index page is still accessible. NGINX is still running.

Let's use Apache Benchmark (ab) to stress the server while it is under slowloris attack

Attack with slowloris : slowloris -s 1000 -v 127.0.0.1

Traffic generation with ab : ab -n 5000 -c 100 http://localhost/



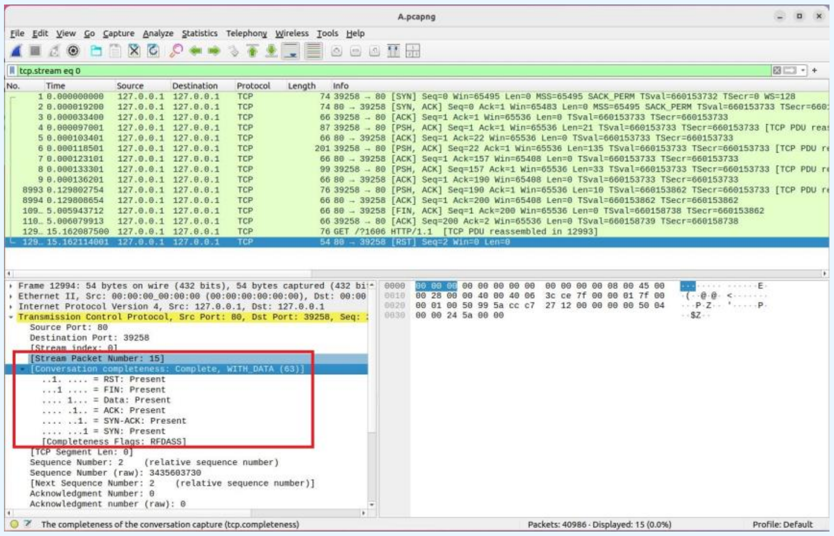
```
codepath@lab000000:~$ ab -n 5000 -c 100 http://localhost/
This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 500 requests
Completed 1000 requests
Completed 1500 requests
Completed 2000 requests
Completed 2500 requests
Completed 3000 requests
Completed 3500 requests
Completed 4000 requests
Completed 4500 requests
Completed 5000 requests
Finished 5000 requests


Server Software:        nginx/1.24.0
Server Hostname:        localhost
Server Port:            80

Document Path:          /
Document Length:        2017 bytes

Concurrency Level:      100
Time taken for tests:   0.495 seconds
Complete requests:      5000
Failed requests:        0
Total transferred:      11255000 bytes
HTML transferred:       10085000 bytes
Requests per second:    10097.05 [#/sec] (mean)
Time per request:       9.904 [ms] (mean)
Time per request:       0.099 [ms] (mean, across all concurrent requests)
Transfer rate:          22195.77 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   1.3      2       6
Processing:     1    8   4.1      8      28
Waiting:        0    7   4.2      6      28
Total:          4   10   3.5      9      29

Percentage of the requests served within a certain time (ms)
  50%      9
  66%     10
  75%     11
  80%     12
  90%     15
  95%     18
  98%     20
  99%     20
 100%     29 (longest request)
codepath@lab000000:~$ 
```

As you can see in the ab screenshot, we have 0 failed requests while the server is under attack.

NGINX is running properly. My DoS mitigation rules are working.

Item #3: A detailed explanation of how you know which `.pcap` file is from the vulnerable server, and which is from the server with DoS mitigation set up

A.pcapng


A.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

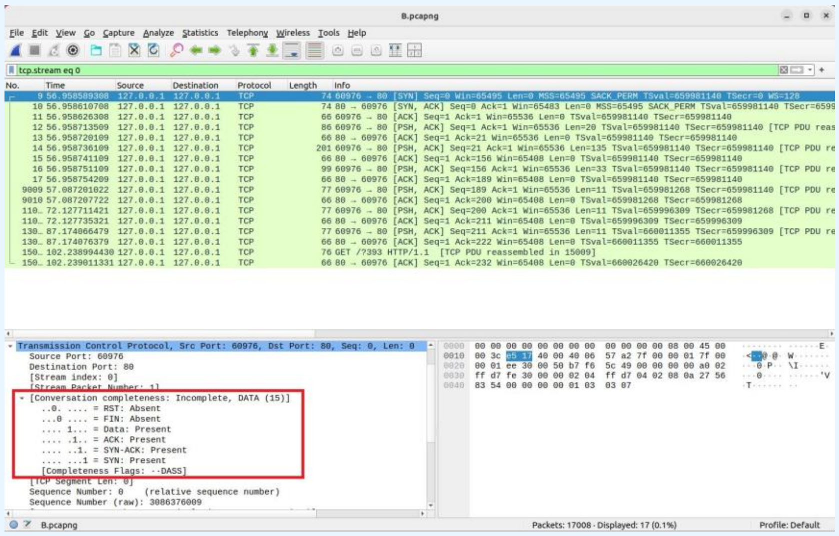
```
tcp.stream eq 0

No.   Time          Source     Destination  Protocol  Length  Info
   1 0.000000000   127.0.0.1  127.0.0.1    TCP        74 39258 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=660153732 TSecr=0 WS=128
   2 0.000019200   127.0.0.1  127.0.0.1    TCP        74 80 → 39258 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=660153733 TSecr=660
   3 0.000033400   127.0.0.1  127.0.0.1    TCP        66 39258 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=660153733 TSecr=660153733
   4 0.000097001   127.0.0.1  127.0.0.1    TCP        87 39258 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=21 TSval=660153733 TSecr=660153733 [TCP PDU rea
   5 0.000103401   127.0.0.1  127.0.0.1    TCP        66 80 → 39258 [ACK] Seq=1 Ack=22 Win=65536 Len=0 TSval=660153733 TSecr=660153733
   6 0.000118501   127.0.0.1  127.0.0.1    TCP       201 39258 → 80 [PSH, ACK] Seq=22 Ack=1 Win=65536 Len=135 TSval=660153733 TSecr=660153733 [TCP PDU r
   7 0.000123101   127.0.0.1  127.0.0.1    TCP        66 80 → 39258 [ACK] Seq=1 Ack=157 Win=65408 Len=0 TSval=660153733 TSecr=660153733
   8 0.000133301   127.0.0.1  127.0.0.1    TCP        99 39258 → 80 [PSH, ACK] Seq=157 Ack=1 Win=65536 Len=33 TSval=660153733 TSecr=660153733 [TCP PDU r
   9 0.000136201   127.0.0.1  127.0.0.1    TCP        66 80 → 39258 [ACK] Seq=1 Ack=190 Win=65408 Len=0 TSval=660153733 TSecr=660153733
8993 0.129802754   127.0.0.1  127.0.0.1    TCP        76 39258 → 80 [PSH, ACK] Seq=190 Ack=1 Win=65536 Len=10 TSval=660153862 TSecr=660153733 [TCP PDU r
8994 0.129808654   127.0.0.1  127.0.0.1    TCP        66 80 → 39258 [ACK] Seq=1 Ack=200 Win=65408 Len=0 TSval=660153862 TSecr=660153862
109… 5.005943712   127.0.0.1  127.0.0.1    TCP        66 80 → 39258 [FIN, ACK] Seq=1 Ack=200 Win=65536 Len=0 TSval=660158738 TSecr=660153862
110… 5.006079913   127.0.0.1  127.0.0.1    TCP        66 39258 → 80 [ACK] Seq=200 Ack=2 Win=65536 Len=0 TSval=660158739 TSecr=660158738
129… 15.162087500  127.0.0.1  127.0.0.1    TCP        76 GET /?1606 HTTP/1.1  [TCP PDU reassembled in 12993]
129… 15.162114001  127.0.0.1  127.0.0.1    TCP        54 80 → 39258 [RST] Seq=2 Win=0 Len=0
```

```
› Frame 12994: 54 bytes on wire (432 bits), 54 bytes captured (432 bi
› Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00
› Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
⌄ Transmission Control Protocol, Src Port: 80, Dst Port: 39258, Seq:
    Source Port: 80
    Destination Port: 39258
    [Stream index: 0]
    [Stream Packet Number: 15]
  ⌄ [Conversation completeness: Complete, WITH_DATA (63)]
      ..1. .... = RST: Present
      ...1 .... = FIN: Present
      .... 1... = Data: Present
      .... .1.. = ACK: Present
      .... ..1. = SYN-ACK: Present
      .... ...1 = SYN: Present
      [Completeness Flags: RFDASS]
    [TCP Segment Len: 0]
    Sequence Number: 2    (relative sequence number)
    Sequence Number (raw): 3435603730
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ··············E·
0010  00 28 00 00 40 00 40 06  3c ce 7f 00 00 01 7f 00   ·(··@·@·<·······
0020  00 01 00 50 99 5a cc c7  27 12 00 00 00 00 50 04   ···P·Z··'·····P·
0030  00 00 24 5a 00 00                                  ··$Z··

The completeness of the conversation capture (tcp.completeness)     Packets: 40986 · Displayed: 15 (0.0%)     Profile: Default
```

The conversation is marked as “Complete, WITH_DATA(63)”
All the control flags are present. That means the process of TCP connexion is completed.
Transmissions going according to plan

“A.pcapng” is from the server with DoS mitigation set up.

B.pcapng


B.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
tcp.stream eq 0

No.   Time           Source     Destination  Protocol  Length  Info
   9 56.958589308   127.0.0.1  127.0.0.1    TCP        74 60976 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=659981140 TSecr=0 WS=128
  10 56.958610708   127.0.0.1  127.0.0.1    TCP        74 80 → 60976 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=659981140 TSecr=6599
  11 56.958626308   127.0.0.1  127.0.0.1    TCP        66 60976 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=659981140 TSecr=659981140
  12 56.958713509   127.0.0.1  127.0.0.1    TCP        86 60976 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=659981140 TSecr=659981140 [TCP PDU reas
  13 56.958720109   127.0.0.1  127.0.0.1    TCP        66 80 → 60976 [ACK] Seq=1 Ack=21 Win=65536 Len=0 TSval=659981140 TSecr=659981140
  14 56.958736109   127.0.0.1  127.0.0.1    TCP       201 60976 → 80 [PSH, ACK] Seq=21 Ack=1 Win=65536 Len=135 TSval=659981140 TSecr=659981140 [TCP PDU re
  15 56.958741109   127.0.0.1  127.0.0.1    TCP        66 80 → 60976 [ACK] Seq=1 Ack=156 Win=65408 Len=0 TSval=659981140 TSecr=659981140
  16 56.958751109   127.0.0.1  127.0.0.1    TCP        99 60976 → 80 [PSH, ACK] Seq=156 Ack=1 Win=65536 Len=33 TSval=659981140 TSecr=659981140 [TCP PDU re
  17 56.958754209   127.0.0.1  127.0.0.1    TCP        66 80 → 60976 [ACK] Seq=1 Ack=189 Win=65408 Len=0 TSval=659981140 TSecr=659981140
9009 57.087201022   127.0.0.1  127.0.0.1    TCP        77 60976 → 80 [PSH, ACK] Seq=189 Ack=1 Win=65536 Len=11 TSval=659981268 TSecr=659981140 [TCP PDU re
9010 57.087207722   127.0.0.1  127.0.0.1    TCP        66 80 → 60976 [ACK] Seq=1 Ack=200 Win=65408 Len=0 TSval=659981268 TSecr=659981268
110… 72.127711421   127.0.0.1  127.0.0.1    TCP        77 60976 → 80 [PSH, ACK] Seq=200 Ack=1 Win=65536 Len=11 TSval=659996309 TSecr=659981268 [TCP PDU re
110… 72.127735321   127.0.0.1  127.0.0.1    TCP        66 80 → 60976 [ACK] Seq=1 Ack=211 Win=65408 Len=0 TSval=659996309 TSecr=659996309
130… 87.174066479   127.0.0.1  127.0.0.1    TCP        77 60976 → 80 [PSH, ACK] Seq=211 Ack=1 Win=65536 Len=11 TSval=660011355 TSecr=659996309 [TCP PDU re
130… 87.174076379   127.0.0.1  127.0.0.1    TCP        66 80 → 60976 [ACK] Seq=1 Ack=222 Win=65408 Len=0 TSval=660011355 TSecr=660011355
150… 102.238994430  127.0.0.1  127.0.0.1    TCP        76 GET /?393 HTTP/1.1  [TCP PDU reassembled in 15009]
150… 102.239011331  127.0.0.1  127.0.0.1    TCP        66 80 → 60976 [ACK] Seq=1 Ack=232 Win=65408 Len=0 TSval=660026420 TSecr=660026420
```

```
⌄ Transmission Control Protocol, Src Port: 60976, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 60976
    Destination Port: 80
    [Stream index: 0]
    [Stream Packet Number: 1]
  ⌄ [Conversation completeness: Incomplete, DATA (15)]
      ..0. .... = RST: Absent
      ...0 .... = FIN: Absent
      .... 1... = Data: Present
      .... .1.. = ACK: Present
      .... ..1. = SYN-ACK: Present
      .... ...1 = SYN: Present
      [Completeness Flags: ··DASS]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3086376009

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ··············E·
0010  00 3c e5 17 40 00 40 06  57 a2 7f 00 00 01 7f 00   ·<··@·@·W·······
0020  00 01 ee 30 00 50 b7 f6  5c 49 00 00 00 00 a0 02   ···0·P··\I······
0030  ff d7 fe 30 00 00 02 04  ff d7 04 02 08 0a 27 56   ···0··········'V
0040  83 54 00 00 00 00 01 03  03 07                     ·T········

B.pcapng     Packets: 17008 · Displayed: 17 (0.1%)     Profile: Default
```

The conversation is marked as “Incomplete, DATA(15)”. The value of DATA is **only** 15 bytes, which is *very* small. This can be tied up to slowloris strategy of sending really small portions of the request without ever sending enough to complete the full HTTP request. The connexion is kept alive, consuming server resources over time.

“B.pcapng” is from the vulnerable server and is under attack.

Submission Checklist

👉*Check off each of the features you have completed.* ***You will only be graded on the features you check off.***

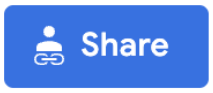
Required Challenges

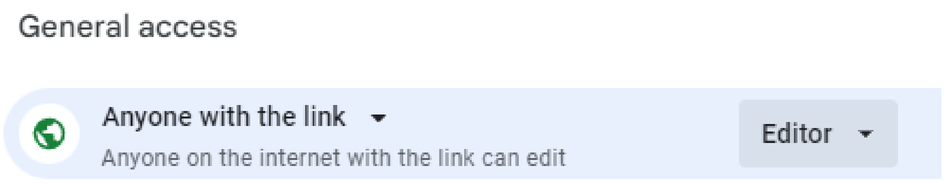
- ~~Item #1~~
- ~~Item #2~~
- ~~Item #3~~

💡***Tip: You can see specific grading information, including points breakdown, by going to 🔗 the grading page on the course portal.***

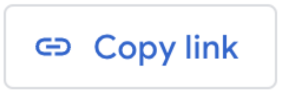
Submit your work!

Step 1: **Click** the Share button at the top of your screen double check that anyone with the link can edit.





Step 2: **Copy** the link to this document.



Step 3: **Submit** the link on the portal.

Grader Comments

Once your project has been assessed, our graders will leave feedback for you in this space. Please do not delete.

Grading Rubric

Reflection Question	Total Received	Total Possible
Reflection Question #1		2
Reflection Question #2		2
TOTAL		**4**
Required Challenges	Total Received	Total Possible
Item #1		3
Item #2		4
Item #3		5
TOTAL		**12**
Total Possible Points		**16**

Grader Feedback